

University of New Hampshire

University of New Hampshire Scholars' Repository

Doctoral Dissertations

Student Scholarship

Winter 2008

The effect of component recognition on flexibility and speech recognition performance in a spoken question answering system

Mike Dalton

University of New Hampshire, Durham

Follow this and additional works at: <https://scholars.unh.edu/dissertation>

Recommended Citation

Dalton, Mike, "The effect of component recognition on flexibility and speech recognition performance in a spoken question answering system" (2008). *Doctoral Dissertations*. 455.

<https://scholars.unh.edu/dissertation/455>

This Dissertation is brought to you for free and open access by the Student Scholarship at University of New Hampshire Scholars' Repository. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of University of New Hampshire Scholars' Repository. For more information, please contact Scholarly.Communication@unh.edu.

**THE EFFECT OF COMPONENT RECOGNITION
ON FLEXIBILITY AND SPEECH RECOGNITION PERFORMANCE
IN A SPOKEN QUESTION ANSWERING SYSTEM**

BY

MIKE DALTON

Baccalaureate Degree (BSEE), UNH, 1999

Master's Degree (MSEE), UNH, 2000

DISSERTATION

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Doctor of Philosophy

in

Engineering: Electrical

December, 2008

UMI Number: 3348311

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]


UMI Microform 3348311

Copyright 2009 by ProQuest LLC.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 E. Eisenhower Parkway
PO Box 1346
Ann Arbor, MI 48106-1346

This dissertation has been examined and approved.



Dissertation Director, W. Thomas Miller III,
Professor of Electrical and Computer
Engineering



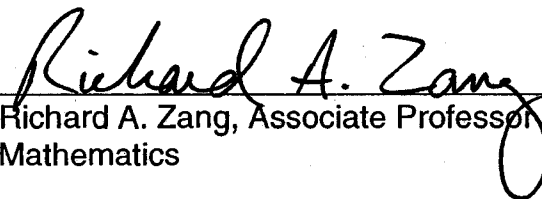
Michael J. Carter, Associate Professor of
Electrical and Computer Engineering



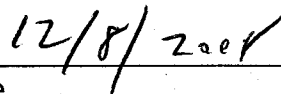
Andrew Kun, Associate Professor of
Electrical and Computer Engineering



William Lenharth, Associate Research
Professor of Electrical and Computer
Engineering



Richard A. Zang, Associate Professor of
Mathematics



Date

TABLE OF CONTENTS

LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF GRAPHS	x
ABSTRACT	xi

CHAPTER	PAGE
I. INTRODUCTION	1
Motivation	2
Spoken Question Answering System Considerations.....	3
Research Objectives	9
Secondary Objective	10
Definitions.....	11
Document Organization.....	13
II. BACKGROUND.....	14
Foundations.....	15
Early Work.....	17
Development	29
Recent Work.....	34
III. MODERN NATURAL LANGUAGE	
QUESTION ANSWERING SYSTEMS.....	41
Introduction.....	41

General Question Answering Approaches.....	42
Closed-Domain Systems.....	45
Indexing.....	46
Weighting Techniques.....	49
Effects of Speech Recognition.....	49
The Current Research.....	51
Summary.....	56
IV. COMPONENT SYSTEM PROCESSING TECHNIQUES.....	58
Processing Techniques.....	58
Weighting Functions.....	60
Comparison of Weighting Functions.....	66
Linear Word Weight Example.....	68
Matching Examples.....	69
V. EXPERIMENTAL DESIGN.....	72
Introduction.....	72
Steps.....	74
Create an SQA Development System.....	74
Develop a specific SQA system.....	78
Set Up a Testing Station.....	79
Optimize the Sample Question Set.....	80
Collect Data.....	81
Analyze the Collected Data.....	82
Conclusion.....	86

VI. SOFTWARE TOOLS.....	87
Introduction.....	87
Sample Questions	88
Grammar Mapping.....	89
Answers.....	91
Conditions	93
Files.....	94
VII. SAMPLE QUESTION SET OPTIMIZATION	95
Phase 0	96
Phase 1	100
Phase 2	110
Summary	111
VIII. DATA COLLECTION.....	112
IX. ANALYSIS	115
Processing.....	115
Analysis	119
X. CONCLUSIONS.....	127
Results of Analysis	128
Software Developed	131
Recommended Use.....	132
Looking Forward.....	133
LIST OF REFERENCES	136
APPENDIX A TEST DATA	144

Test 1	144
Final Test Data	146
Final Summary of Test Data Analysis.....	157
APPENDIX B CD CONTENTS	164
AudioFiles.....	164
DataFiles	165
ExecutableCode	165
SAMSetup	166
SourceCode	167
TestMaterials.....	168
Thesis.....	168
APPENDIX C RELEASE FORM.....	169
APPENDIX D IRB APPROVAL	172
APPENDIX E ACKNOWLEDGEMENT OF FUNDING	174
APPENDIX F SOFTWARE TOOLS.....	175
Using Ted	175
What Ted Does	176
Another Example	178
Opening and Closing Ted.....	179
The Node Properties Dialog Boxes	180
The Menus	183
The Node Questions Dialog Box	188
Questions	189

Boolean Expressions.....	192
Answers.....	196
Using Fred.....	200
The Code.....	204
Set-Up.....	206
File Parsing and Storage.....	207
Utilities.....	208
Response Formulation.....	209

LIST OF TABLES

TABLE	PAGE
1. Words and Linear Weights	68
2. Test 1 Summary	101
3. Frequency of Spoken Queries.....	102
4. Comparison of Systems and Weighting Methods.....	120
5. Comparison of Speech Recognition Performance.....	122
6. Actual Versus Transcribed	123
7. Test 1 Data Summary.....	144
8. Final Test Data	147
9. Data Counts	157

LIST OF FIGURES

FIGURE	PAGE
1. Student Report	4
2. Chomsky Hierarchy	20
3. Question Set 0.....	97
4. Question Set 1.....	99
5. Sample Question Set 2.....	108
6. Ask Fred.....	113
7. Driver Tree	114
8. Data Block.....	115
9. Money Tree Diagram.....	176
10. Student Tree Diagram	177
11. Driver Record Tree Diagram	178
12. Branch Node Properties Dialog Box.....	180
13. Leaf Node Properties Dialog Box	182
14. Lists Dialog Box.....	185
15. Node Questions Dialog Box	189
16. Student Leaf File	201
17. Response Formulation Flow Diagram	214

LIST OF GRAPHS

GRAPH	PAGE
1. Comparison of Normalized Weighting Functions.....	67
2. System Comparison	121
3. Analysis by Subject	123
4. Overlap in System Success Using Both Speech Recognition and Transcriptions.....	126

ABSTRACT

THE EFFECT OF COMPONENT RECOGNITION
ON FLEXIBILITY AND SPEECH RECOGNITION PERFORMANCE
IN A SPOKEN QUESTION ANSWERING SYSTEM

by

Mike Dalton

University of New Hampshire, December, 2008

A spoken question answering system that recognizes questions as full sentences performs well when users ask one of the questions defined. A system that recognizes component words and finds an equivalent defined question might be more flexible, but is likely to have decreased speech recognition performance, leading to a loss in overall system success. The research described in this document compares the advantage in flexibility to the loss in recognition performance when using component recognition.

Questions posed by participants were processed by a system of each type. As expected, the component system made frequent recognition errors while detecting words (word error rate of 31%). In comparison, the full system made fewer errors while detecting full sentences (sentence error rate of 10%). Nevertheless, the component system succeeded in providing proper responses to 76% of the queries posed, while the full system responded properly to only 46%.

Four variations of the traditional tf-idf weighting method were compared as applied to the matching of short text strings (fewer than 10 words). It was found that the general approach was successful in finding matches, and that all four variations compensated for the loss in speech recognition performance to a similar degree. No significant difference due to the variations in weighting was detected in the results.

CHAPTER I

INTRODUCTION

Traditional spoken question answering systems contain a list of specific questions to which the system will respond. The speech recognition engine searches the list of questions, and chooses the specific question most similar to the user's spoken phrase. The system then outputs a corresponding answer.

This research compares such a system to one that uses component word recognition. The component word recognition system contains a list of all the individual words that are used in the original list of questions. The speech recognition engine individually compares each spoken word to those in the list and chooses the most similar. When the entire phrase has been processed, the result of the speech recognition is a string of identified words, rather than a specific question. The system then compares the string of identified words to the original list of questions to determine which question is most similar.

In this study, the comparison of word lists to template questions is accomplished using an adaptation of the vector space model used in Internet search engines. The vector space technique for document retrieval ignores the order of the words in a search query, and instead compares words common to both the query and to the documents on the Internet to identify the specific documents most similar to the query.

Used in a spoken question answering system, the component word recognition system is capable of responding properly to variations of the original questions, so it is more flexible with respect to the questions it can handle. However, the component word recognition system is more likely to make errors in speech recognition because the recognition engine has more options to choose from each time it makes a choice based upon a smaller amount of acoustic information. Because of this, developers of domain-specific spoken question answering systems have shied away from component word recognition in the past.

Motivation

We have come to depend on computers all over the world to store the many bits of information that are crucial to our lives. Businesses, hospitals, and government agencies store enormous quantities of data concerning their daily activities. On the Internet, one can find information on nearly any topic. Our computers contain the answers to many questions.

However, just because information is stored does not mean it can be found when needed. Many techniques of information retrieval have been developed, and are in use today. Public access to information and the desire to automate simple tasks have led to the study of question answering systems, which provide answers to questions worded in a natural language, such as English. These question answering systems include the many search engines that can be found on the Internet. They are generally not domain-specific, and

search through very large amounts of data. In addition, modern question answering systems are usually user initiated, meaning the user starts each exchange.

With the development of speech recognition began the development of spoken question answering systems. These are used today in telephone systems to route calls, gather information, and answer simple questions. Spoken question answering systems are generally domain-specific, and have access to only a small amount of data. They are often system initiated, meaning that the system starts each exchange by prompting the user with acceptable inputs. For example, "Please say the name of the person you wish to speak to".

Spoken Question Answering System Considerations

Suppose a professor has access to a student transcript database system containing reports like the one shown below in Figure 1. A spoken question answering system would allow the professor to make inquiries using a microphone, such as:

What is the student's grade point average?

Who is the student's advisor?

Figure 1. Student Report

```
This is not an official transcript
-----
Name: Jennifer Allen
Address: 402 south main street bivington NH
Date of Birth: 12/21/1987
Student ID: 004-34-7454

Major: Biology
Minor: none
Advisor: John Bosto
Class: sophomore
Status: passing
Credits: 32
GPA: 3.41

Completed Courses

Dept.  CREF  Title                               Grade  ECH
BIOL   403   Introduction to Biology I           A-     14.64
CHEM   405   Chemistry I                         B+     13.32
PHYS   410   Concepts of Physics                 C       8.00
ARTS   426   Introduction to Drawing              A      16.00
BIOL   404   Introduction to Biology II          A      16.00
ANTH   452   Man Through the Years               A      16.00
CHEM   406   Chemistry II                        B      12.00
ENGL   401   English Composition                 B+     13.32

Courses in Progress

Dept.  CREF  Title                               Grade  ECH
BIOL   522   Cellular Processes                  I
BIOL   505   Human Anatomy                       I
CHEM   534   Organic Chemistry                   I
MATH   410   Infinite Mathematics                I
```

A common and reasonable approach is to program the question answering system with complete template questions. A template question is one of the questions to which the system should respond. The template questions are embedded in a grammar file, which is used by the speech recognition engine to recognize spoken inputs. Each time the speech recognition engine is given a phrase, it tries to match it to one of the templates in the grammar file. When a

spoken question matches a template question, an answer is generated that includes data parsed from the record, such as, “the student’s advisor is John Bosto”.

Grammar Rules

To make a system more flexible with respect to the questions it can answer, it is common to incorporate rules into the grammar. A rule is a word in a question that is satisfied by multiple phrases. For example, a rule called <Subject> might match “the student”, “the person”, “he”, or “she”. Then, the question template,

does <Subject> have a major?

would match any of the following.

does **the student** have a major?

does **the person** have a major?

does **he** have a major?

does **she** have a major?

Answer Scripting

For a system to generate natural sounding answers, it must do more than deliver a phrase containing a piece of data from the record. While this approach works fine for some questions, other questions become problematic. Consider the following questions.

Question: what is <Subject> majoring in

Answer: the student is majoring in [Major]

When the system generates an answer, it replaces "[Major]" with data from the record (stored in a node called Major). For the student record shown, the system would answer "the student is majoring in Biology". However, if the student has not yet chosen a major, the system will respond, "the student is majoring in none". This is not a very natural sounding answer. We would prefer something more like "the student has not yet chosen a major", when the contents of the node Major are "none".

Question: has <Subject> failed more than three courses

This question is also problematic. The answer is not contained explicitly in the record, but must be calculated. We want to count the number of entries in the Courses node that have a value of "F" in the corresponding Grade node. We would like to define our answer in this sort of way.

Question: has <Subject> failed more than three courses

Answer: if $\#([\text{Courses}] \text{ where } [\text{Grade}] = \text{F}) > 3$

yes the student has failed $\#([\text{Courses}] \text{ where } [\text{Grade}] = \text{F})$ courses

if $\#([\text{Courses}] \text{ where } [\text{Grade}] = \text{F}) \leq 3$

no the student has only failed $\#([\text{Courses}] \text{ where } [\text{Grade}] = \text{F})$ courses

The scripting language used to specify answers must provide some mathematical functionality. The same scripting language could be used to define conditions for which an answer is chosen, based on the contents of the record.

Question Flexibility

A spoken question answering system such as the one described above will only respond to questions that match one of the template questions. Even if the information is available, if the spoken question is worded differently than the template question, the system is much less likely to respond with a correct answer.

When questions are reworded or worded in an unexpected manner, they still contain most of the same words. Current non-spoken, open-domain question answering systems, such as Internet search engines, use an implementation of the vector space model to compare a query string to documents.

For a spoken question answering system, we wish to compare the spoken input to the template questions. Applying information retrieval techniques to closed-domain spoken question answering systems, a grammar file can be composed of the component words from the template questions. The speech recognition engine is instructed to put together a sentence consisting of the string of individual words in the list that most closely matches the spoken input without regard to sentence structure. The list of component words is then compared to the template questions using an adaptation of the vector space model. Since the

speech recognition engine can put the words in any order, the component recognition technique might correctly match questions with alternate wordings.

Speech Recognition Performance

The current state of speech recognition is such that it performs very well when it has few options in the grammar file, but more poorly when it has many. Thus, a speech recognition engine instructed to recognize either “yes” or “no” will succeed virtually every time. When the recognition engine must choose between 25 template questions, it chooses incorrectly occasionally, even when the spoken input is identical to a template question. When the recognition engine is allowed to create phrases out of component words, it is likely to make considerably more recognition errors.

A system with full question recognition should have reasonably good recognition performance. If a question is phrased properly, the recognition engine will often choose the correct template, resulting in a meaningful answer. However, if the question is worded differently from a template wording, the full question recognition is more likely to fail.

A system with component recognition is likely to have poorer speech recognition performance. Since each word of the spoken input may be matched to any word on the list, the recognition engine makes errors much more frequently. However, if a question is worded in an unexpected manner, but contains many of the same words as a template, the component recognition may

succeed in selecting a logically similar although structurally different question, where full question recognition fails.

Research Objectives

The purpose of this research was to determine if the increase in question flexibility offered by component word recognition could outweigh the decrease in speech recognition performance given the current state of speech recognition technology and an appropriate implementation. This study attempted to quantify the benefit of component word recognition in domain-specific spoken question answering systems using an adaptation of the vector space model. This study also compared four variations of the standard weighting scheme used in vector space based systems to determine how applicable they are to question answering systems in which the target document (a question template) is very short.

The research described in this document progressed as follows.

- A spoken question answering system concerning driving records was created.
- The question template set for the system was created and optimized.
- A number of candidate questions were collected using volunteer test subjects.
- The questions were processed using both full question recognition and component word recognition.
- The results were analyzed and statistics were extracted.

Secondary Objective

A secondary objective of this project was to create a set of tools allowing developers to rapidly create domain-specific spoken question answering systems. The toolset developed is not domain-specific, and contains functionality allowing for both full sentence recognition and component word recognition systems. The development system contains two components; a runtime application and an editor application. The runtime application answers questions about a topic for which it has been configured. The application can support either the full question or component based approaches to spoken question recognition. Spoken answers to sample questions are generated from scripts that can contain conditional responses based on data content.

The editor application is a graphical editor that allows the developer to define the sample questions and appropriate conditional responses required for the runtime application. The editor can create two types of grammar files. The standard grammar file instructs the runtime application to utilize the speech recognition engine for full question recognition. A second grammar file is composed of the component words from the template questions, in support of component based question recognition. In addition, the editor creates a control file containing the conditional scripts used by the runtime application to respond to queries.

Definitions

The following terms are used throughout the document.

Answer – This refers to the scripted answer statement used in developing an SQA system. An answer contains no record data, and may contain mathematical and logical expressions.

Component system – This refers to a SQA system that uses SR to recognize individual query words, and then chooses a fitting sample question.

Editing Application – This refers to the part of the SQA development system described in this document that is used for creating and editing SQA systems.

Full system – This refers to a SQA system that uses SR to recognize entire input queries as sample questions.

IDF – This refers to a commonly used weighting scheme normally defined as the logarithm (base 2) of the so called inverse document frequency function (idf).

When an actual inverse document frequency is used without using the logarithm, it will be referred to as a simple IDF or SIDF.

Query – This refers to the actual spoken phrase uttered by a human user.

Runtime Application – This refers to the part of the SQA development system described in this document that is run as an application to use the SQA system. Although the runtime application does make use of the SR component, the SR component is not considered part of the runtime application.

Sample question – A sample question is a question that has been explicitly entered into the system. A full system SRE recognizes the input as one of the sample questions. A component system SRE recognizes individual words and the runtime application attempts to choose the closest sample question. A sample question is sometimes referred to as a template question.

SQA system – This refers to a spoken question answering system.

SR – This refers to speech recognition.

SRE – This refers to a speech recognition engine, specifically the Microsoft English Recognizer v5.1 recognition engine. The runtime application connects to this engine using the Microsoft Speech Applications Programming Interface (SAPI).

SR Component – This refers to the part of the SQA system that handles speech recognition.

SR response – The speech recognition response is the string returned by the speech recognition component after processing an input query. It is the input to the runtime application.

System Response – A system response is the final output of the system for the user's query. It is an answer to the query that has been evaluated and contains record data if appropriate.

Document Organization

The remainder of this document describes this research in detail.

Chapters 2 and 3 present a review of the literature as it relates to the research. Chapter 4 describes the mathematical models used, and provides justification for their inclusion. Chapter 5 provides an experimental design that describes the steps taken in this research in detail. Chapter 6 is an overview of the software developed for this research, and provides a description of the software functionality. Chapter 7 documents the steps taken to create a useable set of sample questions to be used in the collection of data. Chapters 8 and 9 provide details concerning the collection and analysis of the data. The final chapter draws conclusions based on the analysis.

CHAPTER II

BACKGROUND

This chapter presents an historical overview of basic concepts in natural language processing, providing the foundation for natural language based document retrieval and question answering systems. The subsequent chapter presents a more focused review of contemporary research in natural language based question answering systems.

The field of Natural Language Processing has roots in a number of well-established fields. The most heavily contributing fields are Electrical Engineering, Computer Science, Linguistics, and Psychology. The goals of Natural Language Processing range from applications such as theorem proving and conversational agents to information retrieval and question answering. Due to this variety of contributing fields and applications, relevant research can be found in an enormous number of places. Much of the work done in Natural Language Processing as well as in the contributing fields is not directly related to the problem addressed in this document, yet the work has yielded results which are directly related. Understanding the work in these seemingly unrelated fields is a necessity for future work in Natural Language Processing applications. This section is intended to serve as a summary of the work done in various fields that is now being used in applications similar to the one proposed in this dissertation, or has led to such work.

Foundations

Around 100 BC, Dionysius Thrax of Alexandria wrote a summary of Greek linguistic knowledge. Included in this summary was a description of eight parts of speech; noun, verb, pronoun, preposition, adverb, conjunction, participle, and article. This list of the parts of speech is considered to be the basis of nearly all part of speech descriptions used in every language for the past two thousand years (Jurafsky and Martin, 2000). Thus, Thrax's work is considered the basis of the field of linguistics.

Although man has dreamt for centuries of building a "thinking machine", the first realistic digital computer was designed around the middle of the nineteenth century by Charles Babbage. Babbage's Analytical Engine, as he called it, was entirely mechanical, and used wheels, gears, cogs, and so forth. Babbage spent most of his life trying to construct his Analytical Engine, but failed due to the limitations of the physical system (Tanenbaum, 1992). Nonetheless, this work is considered to be the first real effort towards constructing a digital computer.

In the year 1900, the psychologist Wilhelm Wundt introduced the idea of breaking sentences into constituent parts. These parts could be broken further into smaller constituent parts (Wundt, 1900). For example, a sentence might include a noun phrase, which includes another noun phrase and a prepositional phrase. The prepositional phrase might include a preposition and a noun phrase. Finally, both noun phrases might each consist of an article and a noun. An example is the relatively simple sentence, "The man in the room is hungry." This

method of representing meaning by the use of a hierarchy of constituents later became known as a Phrase Structure Grammar, and is the basis for the Context Free Grammar, which is the most common language theory used in natural language systems.

In 1936, Alan Turing presented a paper to the London Mathematical Society concerning what he called “computable numbers” (Turing, 1936). In this paper, Turing defines the Automatic Computing Machine, which later became known as a Turing Machine. This theoretical machine led to the development of the Finite State Automaton. His work is considered by many to be the foundation of modern Computer Science.

Around the mid 1940’s, many developers including Howard Aiken at Harvard, John Von Neumann at the Institute for Advanced Study in Princeton, J. Presper Eckert and William Mauchly at the University of Pennsylvania, and Konrad Zuse in Germany succeeded in building vacuum tube digital computers. (Tanenbaum, 1992) These machines were quite large and used tens of thousands of tubes. They were difficult to program, expensive to build and maintain, and extremely unreliable by today’s standards. In addition, they were much slower than modern computers, and had a very small storage capacity (around 20 KB). Still, they were digital computers that could be programmed to perform calculations.

In 1948, Claude Shannon first modeled language as a finite state process based on Turing’s work (Shannon, 1948). This effort marks the joining of

language and engineering, and paved the way for much of the work done in this area for the next 50 years.

Early Work

In 1950, Alan Turing considered the question, "Can machines think?" (Turing, 1950). Given the ambiguity inherent in the question, Turing proposed that a new question be considered equivalent, "Are there imaginable digital computers which would do well in the imitation game?" He describes his imitation game as a test in which a human interrogator attempts to distinguish between another human and a digital computer based on a typed conversation. Turing believed that this was possible, but blamed the inability of computers in his day to be successful on their lack of storage capability. In his paper, Turing predicted that by around the year 2000, computers would have a storage capacity of about 125 MB, and he predicted that such a system would be able to fool an interrogator at least 30% of the time (in a five minute interview) on average. This implication that the ability to handle natural language alone is sufficient as evidence of thinking is still controversial today. Yet, it led to the development of many conversational agents and other natural language systems. This was instrumental in the creation of the field known as Computational Linguistics, as well as much of the work described in this paper.

In the mid 1950's, with the development of the transistor computer, researchers began working seriously on the issue of digital computers behaving intelligently. In the summer of 1956, John McCarthy, Marvin Minsky, Claude

Shannon, and Nathaniel Rochester brought together a group of researchers for a two-month workshop on what they decided to call Artificial Intelligence. At that time, natural language systems were mainly based on keyword searches and basic pattern matching.

At the same time, Noam Chomsky published a paper concerning the modeling of language (Chomsky, 1956). In this paper, Chomsky defines a language as the set of sentences it contains. He defines a grammar as a model or mechanism that generates all sentences of a language and no sentences that are not in the language. Equivalently, a grammar can be defined as a mechanism that will determine if a given sentence is or is not part of a language. Thus, the task is to design the grammar for a formal language that accurately models a natural language, or the subset of interest. In his paper, Chomsky formalized three types of grammars (the Finite State Grammar, the Phrase Structure Grammar, and the Transformational Grammar), and compared them in terms of their ability to accurately model the English language. He found that none of these models could serve as models of the English language, but could come close, and they each have more or less ability to be revealing, in that they show some insight as to how natural languages work. Chomsky's Finite State Grammar is based on Turing's Finite State Automaton, and was found to be equivalent to what is now called a Regular Language. Chomsky's second approach, the Phrase Structure Grammar, is a formalization of Wundt's idea of language based on constituent structure. This grammar later became known as the Context Free Grammar, which is the most common grammar type used in

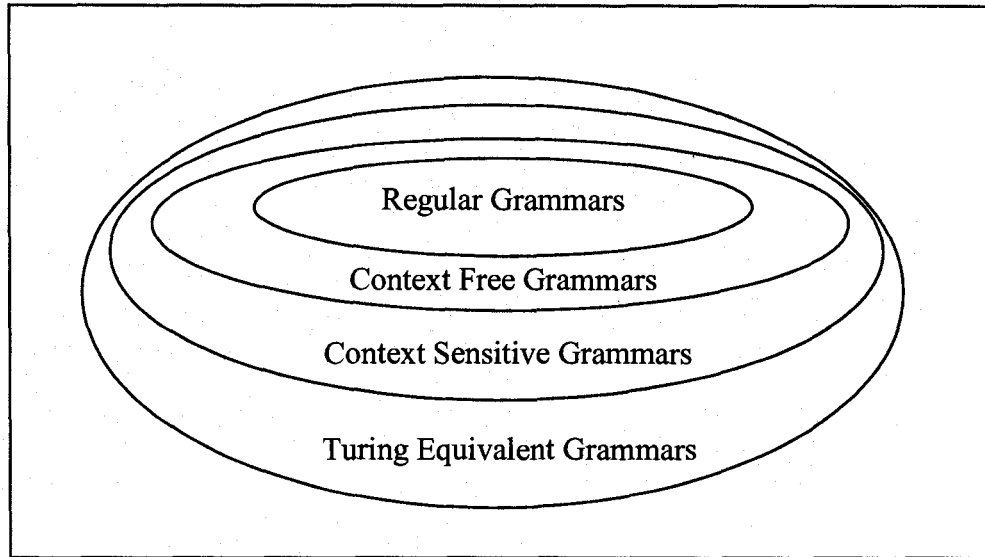
natural language systems today. Finally, he defines the Transformational Grammar, which limits allowable sentences to a small kernel of representative sentences, which can be manipulated by transformations to produce many other valid sentences.

Regardless of the grammar used, a grammar defines a formal language. One type of grammar is considered more powerful than another if it can be used to define languages that the second can not. For example, a context free grammar can define languages that can not be described by any finite state automaton. It is useful to classify specific grammars into groups, or types. These types can be arranged into a hierarchy describing their relationships to each other. That is, less powerful types are considered subsets of the more powerful. The most commonly used is the Chomsky hierarchy (Chomsky, 1959). Chomsky defines four general types of grammars.

- Regular Grammars
- Context Free Grammars
- Context Sensitive Grammars
- Turing Equivalent Grammars

The Venn diagram in figure 2 shows the arrangement of types in the Chomsky hierarchy.

Figure 2. Chomsky Hierarchy



Since a grammar can be defined by a set of rules, a particular grammar is placed in one of these four types based on its rules. A rule (or production) in a grammar shows allowable substitutions of symbols. Each symbol may be a non-terminal symbol (something that has yet to be fully expanded, like a sentence, or a phrase), or a terminal symbol (a word). Non-terminal symbols will be represented by capital letters (A, B, C). Terminal symbols will be represented by lowercase letters (a, b, c). A lowercase x represents a string of terminal symbols of unspecified length. Finally, a Greek letter (α , β , γ) will represent an arbitrary string of terminal and/or non-terminal symbols.

It is important at this point to note that the meaning of the word grammar here is somewhat more general than its popular meaning. Strictly speaking, a

grammar is simply a set of rules or productions. What most people call “grammar” is actually a grammar of language syntax. While natural language processing systems do use grammars for syntax, they also may use grammars for morphology, semantics, spelling, and so on. So, to classify types of grammars according to their rules, the rules are written generally, and need not necessarily apply to syntax or even to language.

Grammars that use more restrictions in their rules are less powerful. The least powerful and most restricted grammars are regular grammars. A regular grammar is equivalent to a regular expression, which is equivalent to a finite state automaton. The rules for a regular grammar are as follows. The left side of the rule must be a single non-terminal symbol. The right side of the rule may include any number of terminal symbols. The rule may contain no more than one non-terminal symbol, and it must appear on the end. (That is, all rules must comply to the same standard. If the non-terminal symbol is allowed on the right end, the grammar is a right linear regular grammar. If the non-terminal symbol is allowed on the left end, the grammar is a left linear regular grammar. For every right linear regular grammar, there is an equivalent left linear regular grammar, and vice-versa.) The following is an example of a regular grammar (specifically, a right linear regular grammar).

$$S \rightarrow aA$$
$$S \rightarrow bB$$
$$A \rightarrow abS$$
$$B \rightarrow bbS$$

$S \rightarrow \epsilon$

Here, ϵ is the null symbol.

Thus, starting with the non-terminal symbol S (Also called the start symbol), we might generate sequences such as:

aab

bbb

aabaab

bbbbbb

aabbbb

bbbaabaab

and so forth. This grammar also could be used to test such sequences. While those given above would all pass, ones such as ababbb would fail.

The general form for a right linear regular grammar is:

$A \rightarrow xB$

A context free grammar is less restricted. The left-hand side must be a single non-terminal symbol. The right hand-side may be any string of terminal and non-terminal symbols. The grammar is "context free" in the sense that the substitution for each non-terminal symbol is independent of what comes before or after it (its context).

In general:

$A \rightarrow \alpha$

The following is an example of a context free grammar.

$$S \rightarrow aAbB$$
$$A \rightarrow aaS$$
$$A \rightarrow Sb$$
$$B \rightarrow abAbS$$
$$S \rightarrow \varepsilon$$

A context sensitive grammar allows more than a single non-terminal symbol on the left-hand side. The grammar is "context sensitive" in the sense that the substitution for each non-terminal symbol may be dependent on what comes before or after it. That is, it substitutes a non-terminal symbol that is in the context of strings of terminal and non-terminal symbols.

In general:

$$\alpha A \beta \rightarrow \alpha \gamma \beta.$$

The rules for a Turing equivalent grammar have no restrictions.

$$\alpha \rightarrow \beta$$

Turing equivalent grammars characterize all languages whose strings can be enumerated by a Turing machine.

In the 1960's and 1970's natural language research concentrated on two major areas; developing new grammar models and taggers, and developing conversational agents. One of the earliest and well known working part of speech taggers was Zelig Harris's Transformations and Discourse Analysis Project (Harris, 1962). This tagger (or parser, as they are often called) worked by checking each word against a dictionary list to find candidates for the correct

part of speech. Then, for each word, a single part of speech tag is chosen from among the candidates using a set of hand written disambiguation rules. In the years following, many approaches were investigated. Stochastic taggers use a training corpus to find the probability of candidate tags in the context of the sentence using Bayesian principles (Stolz, et al., 1965). Another approach investigated was to prune the candidate tags using tests that involved checking suffixes as well as the known tags of the surrounding words, since both suffixes and context can imply a particular part of speech (Klein and Simmons, 1963).

The TAGGIT tagger assigned a part of speech tag for each word using 3300 context frame rules. Each word is checked in the context of a number of words on either side (Greene and Rubin, 1971). This approach differs from that of Klein and Simmons in that the latter only used one word to either side of the word being tagged. In Halliday's Systemic Grammar, inputs are parsed in a way similar to that of a Context Free Grammar, but the words are grouped into clauses and groups (where these words have specific definitions), rather than phrases, which provides more semantic information (Halliday, 1967, 1970). This follows the work of Chomsky in that deriving meaning from the input is a more revealing way to interpret the input.

Many other unique and innovative approaches followed, including Indexed Grammars, which are more powerful than Context Free Grammars and can produce correct sentences that Context Free Grammars can not (Aho, 1969). Other attempts include the Transition Network Grammar (Woods, 1970), The Transition Network Tagger (Johnson, 1983), the Phrase Linking Grammar

(Peters, et al. 1982), and the Lexical Functional Grammar (Kaplan, et al. 1983). Although these grammars offer some useful insight, none have shown themselves to be as useful as the Context Free Grammar.

At the same as time these grammars and taggers were being developed, others were working on conversational agents. A conversational agent is a software entity that interfaces with a user via natural language. Conversational agents are typically limited to some small domain of conversation. One of the earliest, and certainly the best known early conversational agent was Joseph Weizenbaum's ELIZA (Weizenbaum, 1966). The purpose of ELIZA was to study natural language communication between machine and man. ELIZA takes on the role of a Rogerian psychotherapist. As Weizenbaum notes, A Rogerian psychotherapist can maintain a coherent conversation while knowing almost nothing of the real world. Thus, it could be argued that ELIZA had no domain at all. Its purpose was to respond in a natural, though not necessarily useful, way. For its time, the success of ELIZA was somewhat undisputed. While speaking of people who had conversed with ELIZA, Weizenbaum said, "They would often demand to be permitted to converse with the system in private, and would, after conversing with it for a time, insist, in spite of my explanations, that the machine really understood them." (Hofstadter, 1979, page 600) ELIZA worked by using a production system. The system was controlled by scripts, which defined templates. If the input could be matched to a particular template (or frame, as they are sometimes called), the input underwent a series of transformations such as changing "you" to "I" and "are" to "am". Then, the outputs are essentially

canned responses that may involve some of the original input words, some new words, and some transformed words (the use of the word “transformation” is distinct from that used by Chomsky). The following is a small sample of a dialog included in Weizenbaum’s paper.

Input – He says I am depressed much of the time.

Output – I am sorry to hear that you are depressed.

When the input does not match a template, a content free remark is generated such as, “Please go on.” or “What does that suggest to you?”

Essentially, the system looks for particular key words or patterns.

Weizenbaum tried to account for the majority of possible input strings by keeping the templates general. For example,

Input – X no Y. Output – Why not?

Input – X my Y. Output – Why do you say your Y?

Input – Can you X? Output – Perhaps you would like to be able to X yourself.

Input – Everybody X? Output – Can you think of a specific example?

Input – X your Y. Output – Why are you concerned over my Y?

Input – You are X. Output – Would you prefer if I weren’t X?

By thinking about the system’s “domain”, Weizenbaum was able to predict almost all inputs in a general way, and return reasonable outputs.

Several years later, Terry Winograd presented his SHRDLU system (Winograd, 1972). Winograd’s system modeled a world consisting of colored

blocks of different shapes and sizes. The system was able to manipulate these blocks, answer questions about the state of its world, and accept new facts given by the user, such as facts about ownership, support, and proper nouns (names) given to the world elements. It could also reason about why it performed intermediate actions in carrying out a task given by the user. Winograd's system assigns importance to the meaning of the input beyond what can be derived from a simple syntactic analysis. The system uses Halliday's Systemic Grammar, which is discussed above. Winograd defined a method for representing meaning using the PLANNER language (Hewitt, 1971). This method is based on representation of objects, properties, and relations. This makes SHRDLU more flexible for adaptation to other uses than older frame based systems, such as ELIZA. The original implementation of SHRDLU based on the world of blocks behaves impressively.

In 1977, GUS (Genial Understander System) was implemented (Bobrow et al., 1977). GUS was designed to act as a simplified travel agent. Bobrow chose a different path than that of Winograd (although Winograd was part of the GUS team). GUS acts in a way similar to that of ELIZA. The system is template based, and each template has a number of information slots in need of filling. The templates, in this case, may be nested. Handling a particular request involves filling the slots in a tree of templates. The system uses an agenda list to keep track of slots yet to be filled. GUS attempts to fill these slots by asking questions of the user. If the user takes the initiative, GUS will activate an appropriate template, add it to the tree and agenda list, and then try to reclaim

the initiative. GUS uses a Transition Network Grammar. GUS also handles a number of ambiguities related to conversations about making reservations, but these ambiguities are special cases, and their handling is built into the basic design. The authors admit to the simplicity of the system. It is not intended for actual use, but to study language. The system will only make a single trip reservation from Palo Alto California to another city in California.

It is worth noting one other area of research done during these decades. Many researchers realized the need and importance of large corpora, and began collecting them. These corpora generally consist of many samples of text from many sources. A large corpus is useful for a number of things. Most importantly, they are used to test taggers, to test language systems, to develop statistics and rules related to textual information, and to train taggers and other systems that work by statistical methods.

The Brown Corpus is a one million word collection of samples from 500 written texts of American English selected from a variety of genres. It was assembled at Brown University in 1963 and 1964, and is described by Kučera and Francis (Kučera and Francis, 1967). This corpus was tagged mostly by the TAGGIT tagger described above. Words left ambiguous by TAGGIT were hand tagged (Francis, 1979).

The Lancaster-Oslo/Bergen Corpus consists of 500, two-thousand word texts of written British English. It was collected during the 1970's at the Universities of Lancaster, Oslo, and Bergen. The corpus is meant to be a British counterpart to the Brown corpus (Marshall, 1983).

Development

In the last decades of the twentieth century, research in the areas mentioned went mostly along the same paths, but became more complex and specialized. For example, the number of parts of speech used by taggers has grown enormously from Thrax's original eight. Tagsets have been defined to enumerate the parts of speech allowable in the eyes of different researchers. For the most part, these tagsets have grown due to diversification of the basic parts of speech. For example the Penn Treebank Tagset defines separate tags for singular nouns, plural nouns, singular proper nouns, and plural proper nouns. The Penn Treebank Tagset defines a total of 45 word tags (Marcus et al., 1993). The tagged Brown Corpus used 87 distinct tags (Francis, 1979). More recently, the C7 tagset includes 146 word tags (Garside et al., 1997).

Many new theories of grammars have been developed, all having various degrees of power and usefulness in explaining language. A particularly interesting grammar, known as a Tree Adjoining Grammar (Joshi, 1985), is more powerful than a Context Free Grammar. That is, this grammar can generate sentences that are English sentences, but can not be generated by any Context Free Grammar. However, this grammar can not generate all English sentences that can be generated using an Indexed Grammar (Aho, 1969), which can not generate all English sentences that can be generated by every Context Sensitive Grammar. This is typical of new grammar theories. They usually have a power falling somewhere between context free and context sensitive grammars. The

power of a grammar (ability to generate sentences that less powerful grammars can not) is not more important to most developers than the explanatory power of the grammar. That is, it is often desirable to design a grammar that models language in an intuitive way, so as to give some insight to the structure of language itself. The ability for a grammar theory to act intuitively lends to an easier application of the theory, and adds to our knowledge of linguistic structure. In Arnaud Joshi's paper, "Tree adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions?" (Joshi, 1985), the strength of the Tree Adjoining Grammar is explained. The Tree Adjoining Grammar can be used in a natural way to describe dependencies and relations that a Context Free Grammar can not. For example, in the sentence 'The man at the counter is tall.', the word "is" is dependent on the word "man". If "men" were used, the verb would have been "are", not "is". With a Tree Adjoining Grammar, dependencies like this are built into the representation of the grammar. Thus, these dependencies and relations can exist over an unbounded number of words. In a Tree Adjoining Grammar, each sentence is built using basic trees. The main portion of the sentence is defined by a base tree. Dependencies, relations, and redundancies are factored out into auxiliary trees. Sentences are generated (or equivalently, parsed) by adjoining (inserting) auxiliary trees into a base tree. While the auxiliary trees may be adjoined in the middle of a dependency, the relation still holds.

More recent language research has become more specialized and, in many cases, focuses on a particular problem, or construction. An example is,

Kay and Fillmore's 'What's X doing Y? construction'. This work deals with extracting the meaning of sentences of the form "What's this fly doing in my soup?" in spite of the obvious ambiguity which results in the humor of the well known joke (Kay and Fillmore, 1999).

Part of speech taggers have become more reliable and more accessible. Modern taggers use a variety of methods. While some still use production system type rules based on common syntax, others (called stochastic taggers) are trained from pre-tagged corpora and use only statistical information. The advantage to this technique is that the system can properly tag fragments and other improper "sentences" that are used by humans in spite of their grammatical flaws. In 1983, Ian Marshall published a paper describing a stochastic tagger designed to tag the Lancaster-Oslo/Bergen Corpus after being trained using the Brown Corpus (Marshall, 1983). Rather than analyze the sentences in the LOB Corpus syntactically, the system analyzed the tagged Brown Corpus to derive a transition matrix of the probabilities of one tag following another. Marshall's system then generates a list of all possible tag sequences for an input sentence, and using a Bayesian approach, calculates the likelihood of each tag given the preceding tag. Then, the system calculates the total likelihood of each tag sequence to find the most probable. While this approach is more likely to find correct tags for words used improperly, it is of little use to linguists, as it offers no insight as to how language works. The CLAWS tagger works in a similar way, and was also trained using the Brown Corpus (Garside, 1987).

Transformation Taggers use a training corpus to deduce rules to be used in a production system. These taggers have the advantage of being able to deal with incorrect usage, and they provide rules for these usages. It has been known for decades that artificial intelligence systems are good at finding patterns and deducing rules that humans can not. Most importantly, since actual rules are generated, this approach offers more insight to the workings of language. An example of a Transformation Tagger is the Brill Tagger (Brill, 1995).

Another recent change has developed in the collection of corpora. Older corpora consisted of samples of written text. With speech recognition systems becoming more reliable, a number of efforts have been made to collect spoken language corpora. Two well-known examples are the ATIS Corpus (Air Travel Information System), and the Switchboard Corpus. These corpora differ from the traditional ones in that they include such things as false starts, colloquial pronunciations, noise, and extraneous utterances, such as “um” and “uh”. The ATIS Corpus was collected for use in designing automated airline reservation systems, such as the GUS system described above. The samples were collected from volunteers who were led to believe that they were testing an actual working automated reservation system. In actuality, they were conversing with a human in another room (Hemphill et al., 1990). The Switchboard Corpus was gathered in the early 1990's. It contains 3 million words from 2430 telephone conversations (Godfrey et al., 1992).

While theories of language and language processing were being developed by engineers, computer scientists, psychologists, and linguists,

several ideas from the field of information retrieval were formulated that are now finding use in question answering systems, most importantly, the Vector Space Model (Salton, 1971). The Vector Space Model of Information Retrieval is used in many current systems, including most web search engines. (Jurafsky and Martin, 2000) This approach completely ignores syntactic information, and offers no insight to the problems of language. Nevertheless, it has been found to be quite useful in locating documents from a natural language query. The basic idea of the method is that query strings are broken into words, or components. Each component is represented as a vector orthogonal to all others. A resultant vector represents the search query. Each document is represented by a vector in a similar way. Once the vectors are normalized, the distance between the query vector and document vectors serves as a measure of similarity. The approach is made more useful with the addition of term weighting, where certain terms (words) are represented by longer vectors than others. Originally, this weighting was done by hand. Newer approaches use factors such as term frequency, which was actually developed before the Vector Space Model (Luhn, 1957). The idea is simply to give more weight to a term that appears more frequently within a given document. Another factor commonly used in Vector Space Model systems is the inverse document frequency (Sparck Jones, 1972), which essentially penalizes words that are common to many documents, but increases the weight assigned to words that are unique to only a few documents. Most vector space models employ what is called tf-idf weighting (term frequency – inverse document frequency) .

Recent Work

By the turn of the twentieth century, question answering had become recognized as a field of its own. Researchers differentiate between question answering systems on a number of levels. A closed-domain question answering system is designed to answer questions about a particular topic or area. Open-domain systems attempt to answer questions about any topic. The data containing the answers to the questions may be a large or small collection. Most current research has focused on large collection systems, particularly where the collection is the Web. Most Web question answering systems return a document or list of documents. Some return a portion of a document, commonly referred to as a snippet, which contains the answer. A small amount of research has been done on systems that construct answers. Question answering systems can use typed or spoken input.

In 1999, the Text Retrieval Conference (TREC, co-sponsored by the NIST and the US DOD), began its question answering track, allowing developers to compete and compare methodologies. Each year, the conference offers a large collection of text data from newspapers and various agencies, and a list of questions. The set is used to evaluate open-domain, large collection, typed-input question answering systems. Participants test systems that return snippets. The TREC QA track questions and data sets are also used by many developers and researchers who are not participants for system evaluation. "Current Question Answering (QA) systems extract answers from large text collections by (1)

classifying the answer type they expect; (2) using question keywords or patterns associated with questions to identify candidate answer passages; and (3) ranking the candidate answers to decide which passage contains the exact answer."

(Narayanan, 2004, page 1)

A large concentration of current QA research concerns typed, open-domain Web systems. These systems return documents or snippets that answer the question posed. For the most part, they use a vector space model or some combination of vector space, natural language processing, and statistical techniques to compare the words in the search query to words in the Web documents. Some systems that exemplify this approach are given in (Wiegand, 2007), (Radev, 2002), (Roussinov, 2004), and (Padó, 2007).

A system developed at Cornell (Carde, 2000) uses information retrieval techniques (specifically, the SMART Retrieval vector space model system developed by Salton) to generate a list of potentially relevant documents. Then, a shallow semantic analysis is used to find relevant passages within the document, and to form a response.

Researchers working on these systems are generally studying one of two problems: incorrect responses, and the inability of search engines to handle naturally posed questions. "Commercial search portals, such as Google, Yahoo, Alta Vista, and AOL, still lack the ability to answer questions expressed in a natural language." (Roussinov, 2004, page 400)

In closed-domain question answering research, the approaches tend to use templates, or follow the natural language path. The START system (Katz, 1997) allows users to ask questions about a variety of topics using typed natural language. The system converts questions into T-expressions which contain the relevant question information. This expression is then compared to T-expressions in the START knowledge base.

Another example of a current purely natural language approach to closed domain question answering was developed at Rutgers University (Galitsky, 2002). This system was developed for use by financial and legal advisors, where, as the author points out, the information in the database is constantly changing. Question answering is performed by comparing the semantic representation of the query with semantic representations of each of the potential answers.

Other, less traditional approaches include the Microsoft Deep Listener project (Albrecht et al., 1997), which uses a Bayesian approach in an attempt to discern user intentions. The project is based on the ideas of users' goals and beliefs (Horvitz, 2001).

The Proteus Project (Shinyama, 2002) uses the concept of named entities to compare sentences. Essentially, the system uses things that can be named (proper nouns, numbers, and so forth) as key words to determine if two sentences have the same meaning.

There are many facets to the functionality of a QA system, and different researchers have chosen different areas to investigate within the field. Since

most QA systems compare the words in the question or query to the words in documents, there is an assumption made that the answer to the question contains the same words as the question. One way researchers have tried to combat this problem is by using answer checking algorithms that use natural language techniques to determine which candidates properly answer the question. (Narayanan, 2004) (Bilotti, 2007)

Another example can be found in a system developed at the University of Southern California (Hovy et al., 2001). This system uses information retrieval methods to find a number of candidate answers. Then, these candidates are pruned by using a semantic analysis to see if the candidates appropriately answer the question. That is, each question is considered to be of some predefined type. A question that begins in "How many..." should result in a number. A question that starts with "Who...", should result in a name.

A common theme in this development of the next generation of web search engines is the use of the existing redundancy on the Web to generate more reliable answers. Developers work under the assumption that there will be many corroborations and contradictions on the Web. By searching through and comparing multiple sources, the systems attempt to locate popular snippets. A Microsoft project used this technique with idf weighting. (Dumias, 2002) Since then, the idea has become increasingly popular. (Wu, 2007), (Lin 2007)

Another way researchers try to ensure that returned answers fit the posed questions is by making use of existing FAQ (Frequently Asked Questions) pages. The FAQ Finder system (Burke et al., 1997) uses frequently asked questions

pages as a knowledge base. The questions are used as templates and these templates are matched to new questions using a combination of statistical and natural language techniques.

A project at the University of Amsterdam compares typed questions to FAQ pairs by employing a vector space model to determine similarity. For each FAQ pair under consideration, the question, answer, and page title all contribute to the overall weight given. (Jijkoun, 2006)

Yet another FAQ based project, based at the University of Massachusetts, proposes to collect many question-answer pairs from FAQ pages. Questions having the same meaning are linked by comparing all the questions to each other, and by comparing the answers to each other. These linked FAQ pairs can then be used to answer questions at a later time. The research examines a number of common comparison techniques previously found to be successful in conventional open-domain systems. "However, similarity measures developed for documents do not work well for questions because questions are much shorter than documents." (Jeon, 2005, page 617)

The above research proposes to collect information to be used in query responses at a later time. A related idea is being investigated at Google Inc. Since the relevance measure of a page that contains a correct answer may be low, the Google team proposes to collect a large amount of data beforehand. (Paşca, 2007) These projects represent a movement away from searching through documents, and toward focusing on the mechanics of question answering.

As the performance of speech recognition software improved, researchers saw a new way to develop both open-domain and closed-domain QA systems. An early attempt was made by Schofield and Zheng to use speech recognition in an open-domain web QA system. Due to the growing availability of handheld devices, a desire had arisen to develop a hands-free method for question answering. "To our knowledge, automatic answering of spoken natural-language questions has not previously been attempted." (Schofield, 2003, page 178) Schofield and Zheng concluded that "speech can be used for automatic question answering, but that an interface for correcting misrecognitions is probably necessary for acceptable accuracy." (Schofield, 2003, page 180)

Since then, a number of open-domain spoken QA systems have been developed. "In such systems, the automatic speech recognition (ASR) result of the user utterance is matched against a set of target documents using the vector space model, and the documents with high matching scores are presented to the user." (Misu, 2005, page 145) Developers have come to similar conclusions. This combination of an ASR and QA system performs poorly due to the inadequacies of current speech recognition technology. (Harabagiu, 2002)

Early closed-domain spoken QA systems used simple frame approaches. Web Galaxy (Lau, 1997), Jupiter (Zue, 1995), and Dinex (Seneff and Polifroni, 1996), are spoken QA systems that provide information on the World Wide Web or telephone about travel, weather, and dining establishments. These systems all use a basic template approach to answer user queries. The system matches

the query to the closest template, and then fills in the slots to generate a well-defined question.

A team at the NHK Science & Technical Research Laboratories in Tokyo proposed a closed-domain spoken QA system that uses idf weighting and morphology rules. However, for evaluation, the speech recognition was disabled because it was found that the speech recognition performance was too detrimental to the system. (Goto, 2006)

The research described in this dissertation concerns closed-domain spoken QA systems with a small data collection. Successful systems of this type have been template based. Advances have been slow and difficult due to the current state of speech recognition. Vector space models employing tf-idf weighting have been used in many open-domain, typed QA systems, but are not used frequently in closed-domain systems as they have proven ineffective because of the small size of questions. The research in spoken QA systems that does exist has focused on the performance of the question answering algorithms apart from the speech recognition rather than the effect that the speech recognition and question answering algorithms have on each other.

CHAPTER III

MODERN NATURAL LANGUAGE QUESTION ANSWERING SYSTEMS

Introduction

"Current research focuses on text-based, open-domain question answering."

(Mollà, 2007, page 42)

The vast majority of current question answering research focuses on open-domain, text-based systems; specifically web based document retrieval and answer extraction systems. While there are many areas and methods to investigate, one of the most common applications involves the finding of so called "factoids", which are phrases or short excerpts taken from numerous documents that answer a user's query. These QA systems are evaluated and compared annually at workshops held by groups such as the Text REtrieval Conference (TREC) and the NII Test Collection for IR Systems (NTCIR) project.

Although closed-domain QA systems, or Restricted Domain Question Answering (RDQA) systems were examined in past decades due to technical necessity, a renewed interest has surfaced recently for several reasons. New approaches and methods developed for open-domain systems can be applied to closed-domain systems and evaluated. Better performing closed-domain systems can be designed using today's technology. Also there are instances in which the testing of new methods becomes problematic in an open-domain

application. A less complex closed-domain counterpart sometimes serves as a more useful test system.

General Question Answering Approaches

There are three major approaches used in both open and closed-domain systems, and many developers integrate several within a system in various ways.

A Language Model (LM) approach uses linguistic information to extract meaning from text. Generally speaking, terms are tagged as to their part of speech. Text strings are compared by searching for agreement of subject, action, object, and so forth. Since part of the development of such systems entails building the model and choosing appropriate generalized language structures, systems using approaches of this type often investigate specific linguistic question forms, such as why- questions, or what is- questions. LM systems typically use synonym lists, morphology, co-relation, and transformation rules to expand the search query to multiple similar queries. The LM approach is also the commonly used method for transforming a query into a formal representation such as a structured database query (Demner-Fushman, 2005).

A template based approach is used when the query forms are relatively easy to anticipate. Because of this, the template approach has found use mostly in closed-domain systems, where the content is more restricted. A set of template questions, or sample questions, is created that embodies the domain-specific knowledge of the system. Each template has a corresponding output (Sneiders, 2002).

Although the word "template" is commonly used to refer to the target of a search as it is in this case, it more generally refers to a potential phrase that has empty elements, each of which can be filled from a set of predefined values. It is often the case that the word "template" implicitly refers to both sample questions, and questions with empty elements. In open-domain systems, an LM transformation is sometimes used to map a user query to template questions (Katz, 2002).

Perhaps the single most common approach used in question answering is the so called cosine similarity comparison, or vector space model. In mathematics, the cosine of the angle between two vectors is a well known measure of similarity in that it gives the projection of the first vector on the second. That is, it gives the component of the first vector that is common to the second vector. The application of the vector space model is to view each query and sample question as a vector of component words. Although the cosine function is not used computationally, the term "cosine similarity measure" is often used in the literature to refer to the summing of weighted terms approach of the vector space model. See chapter 4 for a detailed explanation of the vector space model. Often called a "bag of words" approach, two strings (usually a query phrase and a target document) are compared by noting words in common. The words are given weights according to rarity. A sum of the weights of the words in common provides a measure of similarity with which any number of such targets can be compared to the query.

Less common approaches include systems that use primarily probabilistic methods (Soricut, 2004), and machine learning techniques (Tsur, 2004).

Since the majority of current QA research is concerned with open-domain systems, the question arises as to the appropriateness of these approaches to closed-domain systems. Closed-domain systems contain domain-specific information, often in the form of template questions with associated responses. The user query is compared to the template questions or answers in some way to determine which question/answer pair is most relevant to the query (Otterbacher, 2004).

In a comparative study, Hidaka and Masui found that the LM approach is more effective than a cosine similarity in finding relevant information when the search target is a document, but that the LM approach was significantly slower (Hidaka, 2003).

However, in an open-domain setting, the query can be quite long, and the search targets are usually documents. A key difference between this and closed-domain systems is that the strings being compared in a closed domain system are typically much shorter. Closed-domain QA researchers have found that although it was once thought that cosine similarity was applicable only to lengthy documents, it works better in some closed-domain systems than an LM approach (Burke, 1997).

(Jeon, 2005) reports that the cosine similarity did not perform as well as a LM approach in a more recent study claiming that the cosine score varies with

template length. Others have addressed this issue by incorporating the template length into the similarity score (Akiba, 2004).

Closed-Domain Systems

Current closed-domain QA systems commonly use some form of cosine similarity measure to compare a user query to templates (Sneiders, 2002) (Hedstrom, 2005).

A related area of research involves answering users' questions by consulting existing Frequently Asked Questions (FAQ) pages on the web. Systems of this type compare a user query to the set of questions and/or answers on one or more pages to find an answer to the query. The target string in this comparison is much more similar in length to a sample question than a text document.

Perhaps the earliest FAQ system is FAQ Finder (Burke, 1997). FAQ Finder is built on a set of explicit assumptions, including that the question part of the QA pair is the most relevant in determining a match between a user query and the QA pair. The FAQ Finder system uses a combination of cosine similarity score and LM comparison. Though both approaches contribute to the success of the system, the team reports that the cosine similarity is the more significant contributor.

Another team including some members of the original FAQ Finder team revisited the project instructing the system to compare the query to the QA answer using LM methods when the query to question comparison was

inconclusive. The results showed that the system did benefit from the additional information for questions of the how- type, as these were the only ones the language model was built for (Mlynarczyk, 2005).

Since the original FAQ Finder system, a number of groups have developed FAQ like systems. A Microsoft project uses various reformulation techniques and a "statistical chunker" to transform the user query into potential answer statement forms for comparison to FAQ answers. They found that transforming a question to an answer representation more often hurts than helps performance, especially for complex questions (Soricut, 2004).

Another approach is to compare the user query to many elements of the FAQ page, including the question, answer, page title, and page text. The reasoning is that questions contained in FAQ files often rely on implicit information. For example, a FAQ page concerning the Ford Mustang automobile might have the question "how much horsepower does the engine have?", without explicitly specifying what engine the question refers to. This system calculates comparisons for a number of combinations and variations of comparisons. It was found that the best performing models used matching based on the question part of the FAQ page (Jijkoun, 2005).

Indexing

Cosine similarity measures generally involve indexing of the strings being compared, assigning weights to the indexed terms, and finally comparing the indexed, weighted terms as vectors. Indexing refers to the choice of terms and

variations to be used in the comparison. There are several common indexing techniques used in open-domain systems, and some have been applied to closed-domain research as well.

Stemming

Word stemming refers to the removal of any affixes present in a word leaving only the root. Thus, "driving", "driver", and "drives" all have the same stem, "drive". Stemming has proven effective in open-domain applications, in which the text being searched may contain multiple forms of a search keyword, suggesting it is more related to the keyword (Crestani, 2001). Some researchers have tried to incorporate stemming into closed-domain systems (Leuski, 2006) (Sneiders, 2002) (Crestani, 2001). However, it has been shown that whether comparing query to question or to answer, word stemming does not aid in finding relevant QA matches (Jijkoun, 2005).

Stop Lists

A stop list is a list of terms that are to be removed as carrying no useful information. Stop lists generally contain words that are common in the language. The removal of stop words greatly increases the performance of open domain systems, where the documents being indexed can be quite large. The use of stop lists has been applied to closed-domain systems as well (Hedstrom, 2005). However, in closed-domain systems, the use of stop lists has not proven effective. It was reasoned that in closed-domain systems, common question

words such as "who" or "how" might be valuable, but are removed because they are common (Crestani, 2001) (Jijkoun, 2005).

N-grams

Another common indexing technique is to group words into multiword terms, or n-grams. This grouping greatly multiplies the computational effort required in the comparison, but provides valuable context information (Otterbacher, 2004). For example, both of questions, "does painting cause headaches?" and, "which painting does he like?" contain the words "does" and "painting", but not in the same order. The part of speech for the word painting is not the same in the two questions, and they have different meanings. Indexing the questions as bi-grams, for example, would create the terms "does painting", and "painting does", which are unique.

An alternative way to retain this contextual information without the added processing associated with n-grams is suggested in a system developed at Google Inc. (Franz, 2002). The system defines "collocations", which occur between two words when the probability of observing the second word is statistically dependent upon the observation of the first word according to the likelihood ratio (Dunning, 1993).

Synonyms

Another technique that demands more processing, but expands the set of index features is the inclusion of synonyms. Synonyms are commonly added

after stemming is done, and before morphological expansion (Burke, 1997) (Sneiders, 2002).

Weighting Techniques

Although phrases are sometimes compared using LM, probabilistic, or machine learning approaches, the most common approach by far within closed-domain systems has been some variation of the cosine similarity approach using classical tf-idf weighting. See chapter 4 for a discussion concerning tf-idf weighting.

There has been little discussion about the use of the term frequency in closed-domain systems. Almost without exception, researchers include the term frequency factor without justification other than its successful history in document retrieval. The logarithm is always used as part of the idf weight, although the base is rarely mentioned, implying a common logarithm as is specified in (Burke, 1997). No substantial efforts have been made to examine the fitness of the inverse document frequency function in closed-domain systems. Researchers feel that while other weighting schemes may prove more effective, the classical idf measure is commonly used because it is arguably the most standard scheme, and has shown success in many applications (Crestani, 2002).

Effects of Speech Recognition

As is the case with typed QA systems, the vast majority of research in Spoken Question Answering (SQA) systems focuses on open-domain problems.

Once again, some of the ideas and findings of open-domain research are relevant to closed-domain efforts.

Studies involving employing SR in spoken document retrieval systems have found that speech recognition errors do not adversely affect the accuracy when the relatively long target documents (audio documents) are converted to text, due to redundancy and contextual information within the document. However, these same studies often suggest that misrecognitions could have a profound effect on system accuracy when the query is being recognized, particularly if the queries are short (Allan, 2002). Allan defines a query as "short" if it has fewer than 30 words.

Some open-domain SQA systems have been developed using Automatic Speech Recognition (ASR), and cosine similarity for document retrieval (Schofield, 2003) (Fujii, 2003) (Akiba, 2004). ASR in this context refers to an SRE that uses a dictation grammar that contains all of the words in the language of interest, rather than an anticipated subset. This combination of an ASR and document retrieval does not perform satisfactorily from a practical point of view (Akiba, 2004). The main problem being speech misrecognitions, suggesting that some mechanism for correcting them be used (Schofield, 2003). Schofield found that when comparing SR inputs to transcribed inputs, the SR errors severely hurt system performance. The system scored 39% versus 58% correct responses for SR and transcribed inputs respectively for one subject, and 26% versus 60% for the other.

Other open-domain developers have stated that SR errors become particularly problematic when the query is short, where "short" may mean anything from 10 to 15 words (Barnett, 2002) to less than 28 words (Crestani, 2001).

Following the trend in current technology, AT&T Corp. was awarded a patent in 2007 for a spoken FAQ type QA system. The system uses classical cosine similarity with tf-idf weighting, which is enhanced by additional language modeling methods (Gupta, 2007).

The Current Research

The research described in this dissertation concerns closed-domain SQA systems. Following the popular methodology for FAQ type systems, sample questions (or template questions), are provided as analogous to the question part of the FAQ QA pair. The user query is compared to each of these sample questions to determine a closest match.

Approach

Given that LM approaches are generally expensive to build and maintain, are processing intensive, and can be at odds with the domain specific aspect of the system (i.e. parts of speech vary, and meanings can become more specific in restricted domains), they are not an attractive choice. In addition, LM approaches tend to increase the size of the system lexicon, decreasing SR performance, which is a major concern in SQA systems.

The cosine similarity is preferred, as it is cheap, fast, and not domain-biased. A cosine similarity measure involving tf-idf weighting is used to compare the user query to the sample questions, based on the belief that the question contains the useful information in matching a query to a QA pair. Template type functionality is offered through the use of grammar rules, but always on a closed-domain system level. That is, the overall approach does not include any specific rules, just the ability for a system developer to add them.

Stemming

Word stemming was not used. The full word offers valuable information concerning parts of speech and context. Consider the questions, "How much does a canoe cost?" and, "Is canoeing safe?" Stemming would remove the fact that the first question is about an object (canoe), and the second refers to an action (canoeing). This information obviously would help in steering the system towards the best sample question.

Stop Lists

Stop lists were not used. The purpose of a stop list is to remove common terms from the query and targets. Since the terms are already weighted based on their actual rarity in the application, further removal based on open-domain generalizations is not needed, and is likely to remove useful words, as discussed above. While a word in a stop list may be common in the language, it may in fact be very rare in the sample question set.

N-grams

Although n-grams have proven useful in capturing context, they also greatly increase the processing time required for each exchange. No attempt was made in this research to find useful n-grams automatically. The system allows for the inclusion of anticipated n-grams to a particular closed-domain system by the use of grammar rules. A rule with multiword elements is treated as an n-gram and given a single weight.

Synonyms

The SRE can only recognize words included in the SR grammar file. The inclusion of synonyms and word variants created by stemming and morphological rules associated with synonym use would require an unacceptable lexicon size without much expected benefit. Again, anticipated synonyms can be added to a particular domain-specific system as a rule.

Similarity Measures

As will be discussed in detail in Chapter 4, several variations of the classical tf-idf weighting are used in this study.

traditional IDF $w_{kidf} = \ln\left(1,000,000 \times \frac{N}{n_k}\right)$

Simple IDF $w_{ksidf} = 4 \times \frac{N}{n_k}$

Linear $w_{klin} = 100 \times \frac{N - n_k}{N - 1}$

Binary $w_{kbin} = 1$

Where N indicates the total number of template questions and n_k indicates the number of questions in which word k occurs. These four cosine similarity functions are intended to evaluate the effect of placing relatively more or less weight on uncommon words, (uncommon within the sample question set).

A natural logarithm was chosen for the first function. In all cases, the system used binary term frequency weighting. If the query word appeared in the sample question, the term frequency is 1. Otherwise it is 0. A binary term frequency was used for two reasons. The term frequency is dependent upon the target document, or sample questions in this case. Thus, weights must be calculated for words independently for each target. This is an undesirable requirement, particularly for systems that update their information frequently.

More importantly, queries and sample questions rarely duplicate words. In a document retrieval task, if a term appears many times within a document, it stands to reason that the document is highly related to that word. On the other hand, in a query to question mapping, in the rare case that a term appears twice, the duplication may have no importance. Consider the question, "What is the color of the book?" Since the word "the" appears twice, its doubled term frequency doubles the weight of the word for that question alone, although the word "the" is no more important than it is in the question, "What color is the book?" In addition, if a sample question worded this way is weighted with a non-binary term frequency, all other queries containing the word "the" would be unfairly biased toward the question with the duplicate word.

Speech Recognition

Almost without exception, developers of SQA systems chose to use ASR, in which the SRE is instructed to recognize words from the application language using a large vocabulary. This is understandable in open-domain settings where the query content is unknown. As shown above, even with LM optimizations, the success of open-domain SQA has been limited. As suggested earlier, this sort of difficulty is one of the motivations for the renewed interest in closed-domain systems. As discussed above SR performance becomes increasingly important as the string length (query length) becomes shorter, and many report 30 words as a cutoff point.

The queries in this dissertation tended to be between 1 and 10 words in length. To achieve acceptable SR performance, ASR was not used. The grammar file contains only words that appear in the sample questions.

In addition to the promise of improved SR performance, there is a more important motivation for not using ASR. Only the words that appear in the sample questions have defined weights. Other words that are recognized by the SRE will be ignored in the comparison, and so there is no benefit to including them in the speech recognition.

Summary

While other studies have explored the use of cosine similarity scores (using weighted sums) to compare short text strings, past research has not addressed the issue of the impact of speech recognition as it applies to such systems, or to closed-domain SQA systems in general. It has been suggested that recognition errors are compensated for when the target document is large. However, the impact of recognition errors on short queries has not been explored to the extent that it has in this dissertation.

In systems that apply cosine similarity scores, classical tf-idf weighting is always used with very little variation. No attempt has been made in past studies to examine weighting schemes other than tf-idf when used in similar applications where short text strings are compared, and specifically in domain-specific SQA systems. This dissertation examined the difference between several cosine similarity weighting methods, and examined the impact of speech recognition

errors on such systems by both comparing the performance to a so called "full" system, and comparing the performance to a system with "perfect" speech recognition by the use of query transcripts.

The system proposed in this dissertation uses a set of template questions to which each user query is compared using several variations of the cosine similarity measure with tf-idf weighting. These variations were compared to see if any showed a significant performance benefit. No indexing techniques, such as stemming, synonym expansion, morphological expansion, n-gram featurng, or stop lists were employed, although the functionality afforded by some of these techniques is embedded in the ability to use grammar rules. The SRE used a grammar containing only words that appear in the template questions, rather than the common large vocabulary ASR.

CHAPTER IV

COMPONENT SYSTEM PROCESSING TECHNIQUES

Processing Techniques

This chapter describes the processing techniques used in the component system. The majority of this processing entails calculating weights for the component words. Four weighting functions were investigated for comparison. These weighting functions are described and evaluated.

The full system uses an SR grammar that contains the sample questions as atomic entities. If the SRE does not find a match with a high enough confidence score, it sends a message to the runtime application specifying that the speech was not recognized. However, assuming that the SRE returns a phrase, it is guaranteed to be one of the sample questions. The SQA runtime application is identical in both the full and component systems. When the SRE returns a recognized query to the full system runtime application, this phrase is compared to the sample questions using the linear weighting method described below. Since the SRE always returns a phrase that is identical to one of the sample questions, the runtime application always finds the same sample question it was given by the SRE. Once the runtime application has chosen a sample question, the corresponding answer can be filled with record data and sent to the user as a system response.

The only difference between the full system and the component system is that the component system uses an SR grammar that contains individual words rather than complete question phrases. The SRE chooses a grammar word for each word in the spoken query and creates a string to contain them. The runtime application receives this string, and compares the string to the sample questions on a word by word basis. This comparison is achieved by using a sum of weights. Although the cosine function is not used computationally, the term "cosine similarity measure" is often used in the literature to refer to the summing of weighted terms approach of the vector space model. The sum of the vector weights is proportional to the cosine of the angle between the vectors as shown by the vector dot product; $A \cdot B = |A||B|\cos(\theta)$, where the dot product is the sum of the vector components.

The sum of weights method for scoring candidate sample (template) questions based on word content assumes that the k_{th} word in the grammar has been assigned a weight w_k . Each sample question is represented by a vector of elements t_{jk} indicating (by 1 or 0) whether or not the j_{th} sample question contains the k_{th} word in the finite grammar. The list of words returned by the SRE in response to a spoken query is represented by a similar word selection vector with elements q_k , which indicate (by 1 or 0) whether or not the spoken query contains the k_{th} word in the finite grammar. Each sample question is assigned a score s_j based on:

$$s_j = \sum_k q_k t_{jk} w_k$$

The sample question with the highest word score is selected as the most likely match to the spoken query.

The operation of this classifier is dependent upon the initial selection of the weights w_k . Four approaches to term weighting were examined.

Weighting Functions

IDF – The logarithm of the inverse document frequency

Linear – A linear mapping of inverse document frequency to weight

SIDF – The literal inverse document frequency function; a simple IDF

Binary – Words are given a weight of 1

IDF

The inverse document frequency weighting method is commonly used in information retrieval. In the literature, idf weighting generally refers to:

$$w_{idf} = \log_2 \left(\frac{N}{n_k} \right)$$

where N is the total number of documents under consideration, and n_k is the number of documents within that set that contain the search word. Although the term N/n_k is the inverse document frequency function, it is common practice to take the logarithm. The base 2 logarithm is consistent with a justification based on information theory. However, both the base of the logarithm and the inclusion of an arbitrary scale factor have no impact on the result when the objective is to compare scores (Robertson 2004, p.503-520), and the natural logarithm was used for this application. The literal inverse document frequency term is

multiplied by one million to give the output a scale similar to the other methods under consideration.

$$w_{kidf} = \ln\left(1,000,000 \times \frac{N}{n_k}\right)$$

Document retrieval systems typically utilize some variation of tf*idf weighting to select the weights which assign relative importance to different words in a query string, where tf is the term frequency. The term frequency is defined as the number of times a search word appears in a document, or in this case, a sample question. This approach is motivated by a statistical model of word occurrence over a large set of independent documents, each containing a large number of words. For a survey of theoretical bases, see (Robertson 2004, p.503-520). While idf weighting may in fact also work well in the sample question selection application, it is not clear that the underlying statistical model is relevant, given the limited number of sample questions, the limited number of words in each sample question, and the likelihood that the sample questions will not be independent.

The idf weights used in this research corresponded to tf*idf weighting with a binary term frequency. The term frequency was deliberately omitted since it was not clear that this was a relevant parameter for selecting compact sample questions, in contrast to selecting many-word documents. However, since individual words did not occur more than once in a given sample question, the two weightings (idf versus tf*idf) were equivalent.

Linear

An alternative approach to selecting the unknown weights w_k is to define mathematically a reasonable performance metric, and then to determine the values for the weights which optimize the performance metric. This approach is used commonly in optimal signal processing, control, and pattern classification applications in which insufficient statistical information is available to use Bayesian optimization techniques. In vector pattern detection applications, it is considered desirable to maximize the distance in the feature vector space between different classes. In the current application, the difference between the sample question scores for the i_{th} and j_{th} sample questions is given by:

$$s_i - s_j = \sum_k q_k (t_{ik} - t_{jk}) w_k$$

where t_{ik} and t_{jk} have values of 0 or 1 signifying the existence of word k in the i_{th} and j_{th} sample questions. If the spoken query is identical to the i_{th} sample question, this becomes:

$$s_i - s_j = \sum_k t_{ik} (t_{ik} - t_{jk}) w_k \quad (\text{where } q_k = t_{ik})$$

A reasonable measure of the overall separation between the scores for correct versus incorrect sample questions is to compute the sum of the differences between the score for the correct template and all scores for incorrect templates, computed over all possible correct templates.

$$s = \sum_i \sum_j (s_i - s_j) = \sum_i \sum_j \sum_k t_{ik} (t_{ik} - t_{jk}) w_k$$

Since the term weight w_k depends only upon k , it can be brought outside the summations for i and j . Distributing t_{ik} , noting that $t_{ik} t_{jk} = t_{jk}$, and then factoring leaves:

$$s = \sum_k w_k \sum_i \sum_j t_{ik} (1 - t_{jk})$$

Since t_{ik} is independent of j , it can be brought outside the third sum:

$$s = \sum_k w_k \sum_i t_{ik} \sum_j (1 - t_{jk})$$

Since t_{ik} has a value of 1 if the word k occurs in the i th sample question, and 0 if it does not, the sum of t_{ik} over all sample questions is equal to the number of sample questions that contain word k . The same holds true for t_{jk} summed over all sample questions. In addition, the number 1 summed over all sample questions yields the total number of sample questions.

$$s = \sum_k n_k (N - n_k) w_k$$

Here, N is the total number of sample questions and n_k is the number of template questions that contain word k . Note that it is not appropriate to compute the sum of the squared scoring differences since the actual decision is based on the linear sum of weights rather than on Euclidean distances in the weight vector space. Also note that it is not necessary to sum the absolute values of the scoring differences since the differences as expressed are always positive. It is generally desirable to select the weights to maximize s . However, the above criterion used alone merely specifies that the weights should be as large as possible.

Another performance criterion that can be considered is the error that will occur in the matching score s_j for a sample question if a speech recognition error occurs relative to word k (either a word was spoken but missed, or a word was not spoken but was falsely detected).

$$|e_{jk}| = t_{jk} w_k$$

It is desirable to minimize the sensitivity of the system to single word speech recognition errors by avoiding over reliance on individual words. In effect it is desirable to spread the significant scoring over as many words in each sample question as possible, while maintaining good separation between the scores for different sample questions. Consider a robustness measure R , which is the summed squared scoring error caused by individual word recognition errors summed over all sample questions and over all words in the grammar.

$$R = \sum_j \sum_k (e_{jk})^2 = \sum_j \sum_k (t_{jk} w_k)^2$$

Again, since t_{jk} has a value of 1 only for words that occur in sample question, the sum of t_{jk} over all sample questions k gives the number of sample questions containing word k , n_k .

$$R = \sum_k n_k (w_k)^2$$

It is desirable to minimize R , with the effect of minimizing the numerical scoring errors that result from speech recognition errors. Note that as the result of the square, this criterion emphasizes reducing larger word error terms more than reducing smaller word error terms. The trend is to equalize the impact of recognition errors across different words.

An overall performance metric can then be defined as:

$$P = c_s S - c_R R$$

where c_S and c_R are constants chosen to emphasize the relative importance of increasing separation between classes versus reducing the impact of word errors, and the negative sign is used so that the optimization goal is to maximize P (tending to maximize s while minimizing R).

The individual word weights w_k can then be chosen to maximize P as follows:

$$\frac{\partial P}{\partial w_k} = (c_S n_k (N - n_k)) - (c_R n_k 2w_k) = 0$$

$$w_k = \frac{c_S (N - n_k)}{2c_R} = 0.5 \frac{c_S}{c_R} (N - n_k)$$

The word weights which optimize the defined performance criteria can be seen to be linearly proportional to the number of sample questions that do not contain the word.

$$w_k \propto (N - n_k) \quad \text{where } 1 \leq n_k \leq N$$

The constant of proportionality is determined by the relative importance assigned to the two individual performance criteria. However, since the final selection of the most likely matching sample question involves simply comparing the magnitudes of the individual sample question scores, the constant of proportionality has no impact on the sample question selection process. Thus, the significant result is the linear proportionality alone. Any numerically convenient scaling of w_k can be used.

For this application, the weights are computed by:

$$w_{kin} = 100 \times \frac{N - n_k}{N - 1}$$

which assigns a linear weight between 0 and 100 to each word. Words that appear in only one question are given a weight of 100. Those that appear in all questions are hypothetically given a weight of 0.

SIDF

The simple idf method was included for comparison. It is literally the inverse document frequency function, rather than the log of such, to which the term IDF more commonly refers. The inverse document frequency function is defined as:

$$idf = \frac{N}{n_k}$$

To scale the function output so that it is more comparable to the other methods, the inverse document frequency function is multiplied by four in this application.

$$w_{ksidf} = 4 \times \frac{N}{n_k}$$

Binary

The binary weighting method was included for comparison. This weighting assigns the same weight to all words, without regard to their frequency of occurrence in the sample question set. The technique was included in order to test the hypothesis that question frequency information is important, and thus the loss of that information is likely to result in poorer performance.

$$w_{kbin} = 1$$

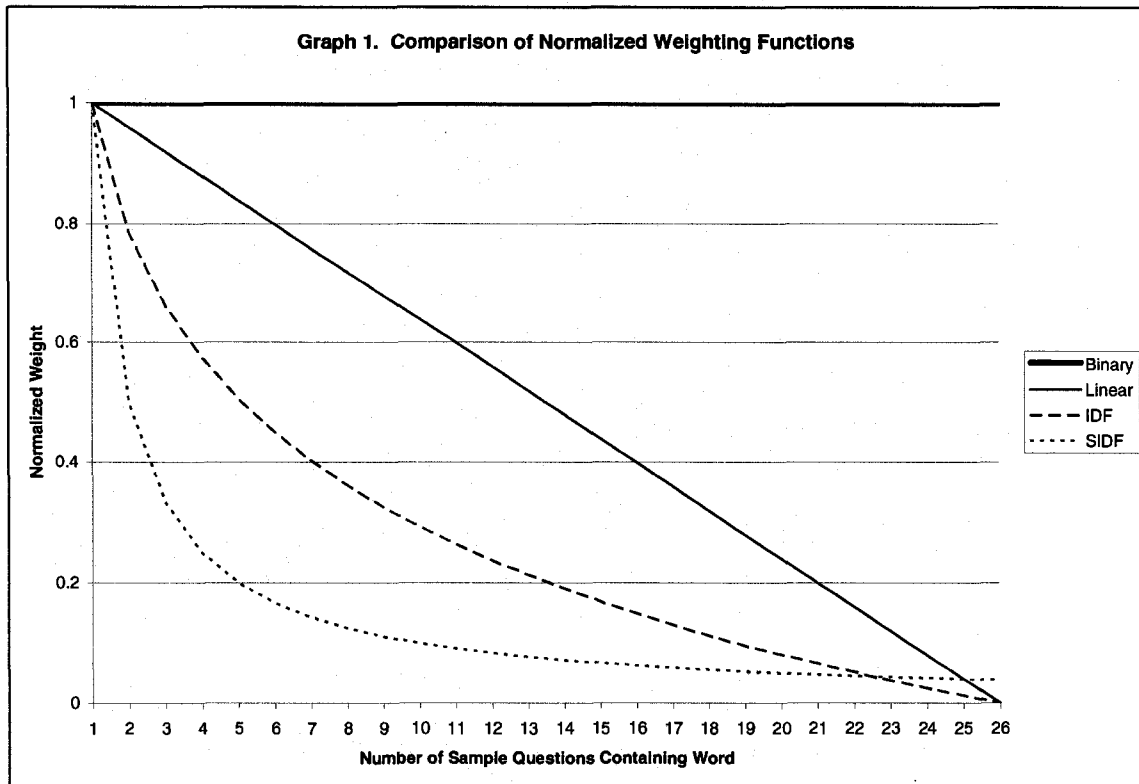
The score for a sample question in this case is equivalent to a count of the number of words shared in common between the sample question and the SR response to the spoken query.

Comparison of Weighting Functions

Graph 1 below compares the four weighting methods. The x-axis represents the number of sample questions in which a word appears. The y-axis represents the weight given to the word. The functions have been normalized for comparison. Once again, in this application we are comparing two values on the same graph so the magnitude of the values is not important, only the relative values. While the shape of the curve may have an effect on the results, the scaling does not.

Theoretical justification for the IDF and linear methods has been given. Both are reasonable candidates for weighting methods in this application, and both assign more weight to words that are more rare. They differ in one respect. While the linear method applies a weight proportional to the rarity of the word, the more popular IDF method places more emphasis on rarity, giving a higher than proportional weight to rare words, and a lower than proportional weight to common words. This can be seen in Graph 1.

For comparison, two more extreme weighting methods are considered. The SIDF method places a very strong emphasis on word rarity, more than IDF. The binary weighting method places no emphasis on word rarity. All words have an equal weight regardless of their question frequency.



Linear Word Weight Example

To provide an example, the linear weights calculated for this research are given below. The weights were scaled as described above so that a word that appears in only one of the 26 questions, like "status", receives a weight of 100. The word "license" appears in two questions, so it has a weight of 96. The word "many" appears in three questions, so it has a weight of 92. The most common word was "what", which appears in 19 of the questions. It has a weight of 28.

Table 2 shows a list of all words used in the driver record test along with their linear weights.

Table 1. Words and Linear Weights

Word	Weight	Word	Weight	Word	Weight
what	28	much	100	eye	100
weight	96	many	92	does	76
weigh	100	license	96	date	100
type	96	is	40	convictions	92
the	92	in	100	color	88
status	100	how	88	birth	100
social	100	height	100	been	100
security	100	have	80	are	96
restrictions	100	has	100	aliases	100
points	100	hair	96	address	100
of	84	gender	96	accidents	100
number	100	first	100	<Subject>	60
name	92	eyes	100	<PossessiveSubject>	44

Matching Examples

The examples below were taken from the data collected during this research. In each example, the query is compared to each of the sample questions. If a word appears in both the query and a given sample question, the weight for that word is added to the total score for that sample question. Four sample question comparisons are shown for each weighting method.

Linear Weight Example

Spoken Query: how many accidents does the driver have

Component SR Response: how many accidents the driver have

Question 21: how many points does <Subject> have

$$88 \quad 92 \quad 0 \quad 0 \quad 60 \quad 80 = 320$$

Question 23: how many convictions does <Subject> have

$$88 \quad 92 \quad 0 \quad 0 \quad 60 \quad 80 = 320$$

Question 24: what type of convictions does <Subject> have

$$0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 60 \quad 80 = 140$$

Question 25: how many accidents has <Subject> been in

$$88 \quad 92 \quad 100 \quad 0 \quad 60 \quad 0 \quad 0 = 340$$

IDF Weight Example

Spoken Query: how many accidents does the driver have

Component SR Response: how many accidents the driver have

Question 21: how many points does <Subject> have

$$15 \quad 15 \quad 0 \quad 0 \quad 14 \quad 15 = 59$$

Question 23: how many convictions does <Subject> have

$$15 \quad 15 \quad 0 \quad 0 \quad 14 \quad 15 = 59$$

Question 24: what type of convictions does <Subject> have

$$0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 14 \quad 15 = 29$$

Question 25: how many accidents has <Subject> been in

$$15 \quad 15 \quad 17 \quad 0 \quad 14 \quad 0 \quad 0 = 61$$

SIDF Weight Example

Spoken Query: how many accidents does the driver have

Component SR Response: how many accidents the driver have

Question 21: how many points does <Subject> have

$$26 \quad 34 \quad 0 \quad 0 \quad 9 \quad 17 \quad = 86$$

Question 23: how many convictions does <Subject> have

$$26 \quad 34 \quad 0 \quad 0 \quad 9 \quad 17 \quad = 86$$

Question 24: what type of convictions does <Subject> have

$$0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 9 \quad 17 \quad = 26$$

Question 25: how many accidents has <Subject> been in

$$26 \quad 34 \quad 104 \quad 0 \quad 9 \quad 0 \quad 0 \quad = 173$$

Binary Weight Example

Spoken Query: how many accidents does the driver have

Component SR Response: how many accidents the driver have

Question 21: how many points does <Subject> have

$$1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad = 4$$

Question 23: how many convictions does <Subject> have

$$1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad = 4$$

Question 24: what type of convictions does <Subject> have

$$0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad = 2$$

Question 25: how many accidents has <Subject> been in

$$1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad = 4$$

CHAPTER V

EXPERIMENTAL DESIGN

Introduction

The purpose of this research was to compare methods for responding to spoken queries. Two base systems were used in the comparison. The "full system" employed speech recognition to compare a user query to a number of predefined sample questions. The "component system" instructed the SRE to recognize words individually, and then used processing techniques to compare the SR response to the same sample questions used in the full system.

It was assumed that the component system would suffer a loss in SR performance due to the larger number and smaller size of grammar candidates. This assumption had to be tested.

It was hypothesized that the component system would be more flexible than the full system in that it would succeed in generating a "proper response" to a greater variety of "reasonable questions" than the full system. This hypothesis had to be tested.

Further, it was hypothesized that the benefit gained by the flexibility of the component system would outweigh the relative loss in SR performance as compared to the full system. That is, the advantages gained would more than compensate for the loss incurred, and the component system would be more

successful in producing proper responses to reasonable questions than a corresponding full system when questioned by random untrained users. This hypothesis had to be tested as well.

In addition to the base system comparison, four different component system processing techniques were evaluated. The IDF technique is based on $tf \cdot idf$ weighting, which has been very successful in performing non-spoken tasks similar to the one proposed. The linear weighting method tends to reduce the sentence error caused by any one SR misrecognition, while still emphasizing rare words over common words. Two other techniques (SIDF and binary) represent two extremes that bracket the first two techniques. See Chapter 4, Component System Processing Techniques for more details concerning these four methods. Another objective of this research was to examine and compare the success of these processing techniques to see which ones might apply to the current application.

It should be clear that the goal was to show that a component system can outperform a full system by providing a successful example. No claim is made that component systems will have superior performance to corresponding full systems in all scenarios. The ability of either type of system to respond properly is a function of the implementation and intended application of the systems.

It should also be noted that this comparison is dependent upon the current state of SR technology. In past years, SR technology lacked the performance needed for component type systems. It is expected that in the future, speech recognition will improve to a point such that the undesired effects are negligible.

This study compared systems in the context of today's SR performance. However, some insight into the impact of SR performance was obtained by duplicating the analysis using human transcriptions of the spoken queries in place of the SR output. This was equivalent to using an error free SR system.

Steps

The research described in this document consisted of the following steps:

1. Create an SQA development system
2. Develop a specific SQA system for testing
3. Set up a testing station
4. Optimize the sample question set
5. Collect data
6. Analyze the collected data

These steps are described in detail in subsequent chapters. However, they are summarized together in this chapter in order to give a concise overview of the research performed.

Create an SQA Development System

The first step in the process was to create an SQA development system.

The following issues were considered:

- Final test platform
- Editor portability

- Domain independence
- System comparability
- Modern features
- Data logging
- Input comparability(Internal validity)

Final Test Platform

The runtime application used in this research was expected to be compatible with an existing speech-controlled project. This existing project was written in C/C++, and uses the Microsoft English Recognizer v5.1 recognition engine. It connects to the SRE using the Microsoft Speech Applications Programming Interface (SAPI). Therefore, the runtime application component of the development system was written in C, and uses the same connection functions as the existing project.

Editor Portability

The creation of an SQA system need not necessarily be done on the same machine that the final SQA system will be run on. The Java programming language was chosen for the development system's editing functionality due to its platform independent nature. Thus, the creation and use functionalities of the SQA development system were separated into two components referred to as the editor application and the runtime application.

Domain Independence

Although any given SQA system developed using these tools will be domain specific, the development tools themselves must not be. To provide a fair comparison, the development system was not biased toward any particular domain. It does not contain any built in information, such as stop lists, synonym lists, or named entities. Any SQA system developed uses only the information entered using the editor for that particular system.

System Comparability

The purpose of this research was to compare systems that differ in only one respect; full sentence versus component word grammars. It was essential that the full and component systems have the same features and implementation. This was achieved by using the same editor and runtime applications for both systems. The only difference between the two systems is the SR grammar file that is generated by the editor application. Thus, the two types of systems are developed in parallel, and will contain the same data, including sample questions, answers, and features.

Modern Features

Modern SQA systems have certain features expected by developers. To test the hypotheses put forth in a realistic way by today's standards, the following features were included.

- Developers can define synonym lists (grammar rules) such that any item on the list will be recognized as that list.
- Answers are scripted. Data from a designated source are fetched and inserted into the answer script at runtime.
- Answer scripts can include data item counts and comparisons.
- Answer scripts can include basic arithmetic and Boolean operations.
- Conditions can be associated with answer scripts such that a particular answer script is only output if the condition it met.

Data Logging

The runtime application was designed to store information during test runs. Several types of log files were written as the system ran. The system also stored each spoken query as an audio file for additional processing at a later time. The application stored a trial number in a file as well. This number was incremented with each new subject to ensure that each subject was uniquely identifiable.

Input Comparability (Internal Validity)

It was important that the inputs given to both systems were very similar in order to conclude that any differences in system success were based on the differences in the systems, not the input data. As subjects posed queries, the phrases were recorded as audio files. These files were processed by systems of each type to ensure that all systems were given identical input data.

Develop a Specific SQA System

The domain chosen for testing and data collection was driver records. A corpus of "scrambled" driver records was obtained from a law enforcement agency. The records were scrambled in that all of the data entries (first name, last name, dates, etc) had been randomly shuffled between records. While the resulting records contained realistic information, in a real format, no information about real drivers was retained. The records were in the form of formatted text file results to a database query.

Study of the sample records led to a generalization of the driver record structure using all possible fields, which was depicted in the editor application. A parser program was written to read a sample record from the driver record file and store the information in a format specific to this development system.

Part of developing an SQA system is choosing a sample question set that will represent a large proportion of the queries users will pose. Assuming that any sample question should result in a response containing an item (one or more) from the record, a question was written for each piece of data a user might inquire about. For some pieces of data, several question phrasings were used. This set of sample questions is referred to as question set 0, and contains 25 sample questions.

Set Up a Testing Station

Since the question set optimization step required the gathering of information for subjects using the systems, a test station was required before continuing. The testing required a computer to run the runtime application. A Dell Latitude D610 was chosen for convenience. The station also required a microphone, speakers, and a mouse. Standard inexpensive devices were purchased from a local department store. The computer system was positioned on a desk so that the mouse was within reach of the subject, and the microphone was facing the subject.

The participants required some instruction as to what they should do to test the system. This is somewhat problematic since any suggestions toward phrasing are likely to bias the subject, and this study concerns the phrasing of queries. The goal was to gather as wide a variety of queries as possible. Ideally, some should match the sample questions exactly, others should not match but be reasonable queries, and some should be queries that are not reasonable for such a system to answer. To offer the participants enough information to use the system, two testing materials were made.

An instruction sheet explained that the system answers spoken questions, and that the domain is driver records. It also described the operation of the system, including which mouse button to click, when to speak, and so forth.

A second sheet showed a tree diagram of the driver record. The tree showed node names corresponding to table column names for the available data. This diagram was altered over the optimization process as leaves that did

not correspond to any sample questions were removed. The final testing materials are shown in Chapter 8, Data Collection.

Optimize the Sample Question Set

The next step in preparing the system for testing was to optimize the sample question set. The success of a QA system is generally very dependent upon the sample questions provided. It is reasonable to assume that either type of system will succeed more frequently if a greater number of user queries are anticipated and represented in the question set. It is also reasonable to assume that increasing the size of the question set will decrease the SR performance of either system. Therefore sample questions that are never used are detrimental to the system.

The question set was optimized in three phases or iterations. In each phase, subjects provided queries to the system. These queries were recorded and analyzed. Using this analysis, the question set was modified by removing unused sample questions, and adding new questions. The details of this process are described in Chapter 7, Question Set Optimization, and are summarized here.

In phase 0, question set 0 as described above was tested using five participants. An analysis of the queries posed led to the addition and removal of a number of questions resulting in question set 1, which contains 39 sample questions.

In phase 1, question set 1 was tested using five new participants. A deeper analysis of the queries was performed resulting in a list of reasons for failure and frequency, and a list of all questions asked and frequency. The list of questions was also grouped by associated answer to determine which pieces of data were requested most often. This analysis resulted in question set 2, which has 26 sample questions that correspond to 83% of the total questions asked during phase 1.

To verify this modification, the recorded queries of phase 1 were reprocessed using question set 2. The success of both systems improved significantly compared to the question set 1 test.

In phase 2, question set 2 was tested using a new group of five participants. Both systems performed acceptably, and the data collected in this phase were used in the final analysis.

Collect Data

The target population for this study was average native English speaking people who had no prior experience or training with this particular SQA system. The subjects used in the study were college students who were taking at least one computer science course because these subjects were available. The sample included a range of ages (from 18 to 36) and both male and female participants, although the majority was male.

The subjects were given the testing materials and asked to sit in the testing station chair. Subjects were given no additional instruction concerning

the phrasing of queries. Subjects were not told how many queries to pose. Some subjects are likely to think of fewer queries than others. An imposed number of queries might force these subjects to create new questions in an unnatural way, biasing the experiment.

Each subject was left alone in a room with a closed door so they would be less likely to feel awkward. The subjects exited the room to signal completion of the testing. Data were gathered from an additional 15 subjects, for a total of 20 subjects to be used in the final data analysis.

Analyze the Collected Data

As stated in the introduction of this chapter, the objectives of this study were to:

- Test the hypothesis that a component system would succeed in generating a "proper response" to a greater variety of "reasonable questions" than the full system.
- Test the assumption that the component system would suffer a loss in SR performance as compared to the full system due to the larger number and smaller size of grammar candidates.
- Test the hypothesis that the benefit gained by the flexibility of the component system would result in the component system being more successful in producing proper responses to reasonable questions than a corresponding full system when questioned by random untrained users.

- Examine and compare the success of four different processing techniques to see which ones might apply to the current application.

To accomplish this, the analysis results are organized into four sections.

- Comparison of systems and weighting methods
- Impact of speech recognition
- Analysis by subject
- Overlap

Comparison of Systems and Weighting Methods

The purpose of this section is to provide an overall evaluation of the systems tested, including the component system with each processing technique. The systems were also compared to test the hypothesis that the component system would be more successful than the corresponding full system, and to compare the processing techniques.

The systems were evaluated, and the following values were reported.

- The total number of reasonable queries
- The number of reasonable queries each system responded properly to
- The percentage of reasonable queries each system responded properly to

To obtain these values, some measure was needed to objectively determine which queries were reasonable, and which responses were proper. In

general, a query was considered to be reasonable (or equivalently to have a reasonable matching sample question) if it was "fair" to expect the system to answer the question using the information available to it. The guidelines developed were as follows:

A query was considered reasonable if the following were all true:

- The query elicited information that was contained in the record.
- A sample question existed that returned the requested information.
- The key words in the query were contained in the SR grammar file.

Where the key words are the domain specific words that normally refer to a piece of information, such as points, address, or convictions.

A response was considered a proper response if it answered the user's question in a satisfactory and expected way. A more detailed discussion of these criteria is given in Chapter 9, Analysis.

The data collected were analyzed to compare the systems. Margins of error were calculated to determine which differences were significant.

Impact of Speech Recognition

The purpose of this section is to provide data consistent with the hypothesis that the component system would suffer a loss in speech recognition performance. This section contains two parts. The first part is a comparison of

speech recognition between the full system and the linear component system. For each system, the total number of fair inputs was found, and the number of correct recognitions was determined.

The total number of fair inputs to the full system is the number of queries that exactly matched a sample question. The sample questions are the inputs the SRE was instructed to recognize. The recognition was considered correct if the phrase chosen by the SRE was the same as the spoken query.

The total number of fair inputs to the component system is the number of words that were uttered and appeared in the component system's SR grammar. Again, these words are the inputs the SRE was instructed to recognize. The recognition was considered correct if the word returned by the SRE was the same as the word that was spoken. Using these numbers, the percentage of correct recognitions was calculated and compared for the two systems.

The second part describes simulated "perfect" speech recognition. The linear component system was used to process transcribed text from the test queries. The performance of the system was compared to that of the same system using real speech recognition.

Analysis by Subject

The purpose of this section is to examine the effect caused by variations between test subjects to determine how consistent the system performance was

across subjects. It also considers the possibility of correlations between overall system performance and specific characteristics of the test queries.

The analysis shows the percentage of queries responded to properly by both the full and linear component systems for each of the 20 test subjects. In addition, the spoken queries were categorized and these query categories were examined as they relate to the success of the two systems. The percentage of participants who benefited significantly from the component system was calculated.

Overlap

The purpose of this section is to show the extent to which the systems agreed, and disagreed. Using the data from the full and linear component systems tests, the following quantities were determined:

- The number of queries the full system responded to properly, but the linear component system did not
- The number of queries the linear component system responded to properly, but the full system did not
- The number of queries both systems responded to properly
- The number of queries neither system responded to properly

Conclusion

The design described was implemented and provided data and analysis sufficient to test the assumptions and hypotheses stated.

CHAPTER VI

SOFTWARE TOOLS

Introduction

The software tools developed for this research together comprise a complete SQA development system. The system contains two parts; an editor component, and a runtime component. For a complete explanation concerning the use of these tools, see Appendix F, Use of Software Tools.

The focus of this research was on spoken question identification. However, a fully functional SQA system must generate appropriate spoken responses as well. A secondary goal of the project was to utilize the spoken question identification capability as the front end to a complete SQA system. The development of a fully functional system defined by coupled question and answer scripts served both to demonstrate the validity of the format for defining sample questions as used in the research, and to provide a platform for future research using complete SQA systems.

The development system allows a developer to create new SQA systems by defining a set of sample questions and corresponding answers. Once the question/answer pairs have been defined, the developer can choose to create a full system, or a component system.

Sample Questions

The sample questions control the domain specific behavior of the SQA system in both the full and component types. They provide all of the domain specific information the system uses to respond to user queries. Sample questions are written in a scripting language developed for this research. Sample questions may contain words and rules. A rule corresponds to a rule in the grammar file. A rule is simply a placeholder that is associated with several options. During recognition, the SRE will recognize any of these options as acceptable matches for the rule.

For example, consider the rule <Subject>, which matches any of the following:

he

she

the driver

the operator

Using the <Subject> rule, we can define questions such as these.

Question: how many points does <Subject> have

Question: does <Subject> have ... convictions

During recognition, the SRE will consider the following for matching to the first sample question:

how many points does he have

how many points does she have

how many points does the driver have

how many points does the operator have

Note that rules are placed inside chevrons (<>). Also note that an ellipsis (...) may be used to represent a filler model, which will match any extra or junk words.

Grammar Mapping

Once the sample questions have been entered, the editor application can be used to generate a grammar file. The two questions above would be represented in a full system grammar file as shown below.

[<Start>]

<Start> = how "how " many "many " points "points " does "does " "{9 " <Subject>
"} " have "have "

<Start> = does "does " "{10 " <Subject> " } " have "have " ... "..." convictions
"convictions "

[<Subject>]

<Subject> = he "he "

<Subject> = she "she "

<Subject> = the driver "the driver "

<Subject> = the operator "the operator "

The SRE is instructed to compare each spoken query to the two sample questions. Either sample question might trigger the <Subject> rule.

The component system grammar file would look like this:

[<Start>]

<Start> = <Sentence>

[<Sentence>]

<Sentence> = <Word> <Sentence>

<Sentence> = <Word>

[<Word>]

<Word> = convictions "convictions "

<Word> = does "does "

<Word> = have "have "

<Word> = how "how "

<Word> = many "many "

<Word> = points "points "

<Word> = ... "... "

<Word> = <Subject>

[<Subject>]

<Subject> = he "he "

<Subject> = she "she "

<Subject> = the driver "the driver "

<Subject> = the operator "the operator "

In this case, the SRE is instructed to build a phrase using words from the sample questions. Note that the grammar file contains only words that are used in the sample questions. Other words will generally be misrecognized as one of these words.

Answers

The answer scripting is somewhat more complicated since answers can contain record data and mathematical functions. Record data are specified using brackets and a number which identifies an information field in the record.

Question: how many points does <Subject> have

Answer: the driver has [35] points

If the field contains a single value, it is inserted into the answer statement when the system responds.

Response: the driver has 5 points

If the field has multiple values, they will all be listed.

Question: where has <Subject> had accidents

Answer: the driver has had accidents in [64]

Response: the driver has had accidents in Concord Lee Durham

Arithmetic operators (+, -, *, /) may be used between static numbers and/or record data items. The expressions are evaluated when the system responds.

Question: how many points does <Subject> have

Answer: the driver has [35] + [36] + [37] points

Response: the driver has 12 points

Fields that contain multiple values can be handled with filters. A filter returns only the values that meet the filter criteria.

Question: where has <Subject> had fatal accidents

Answer: the driver has had fatal accidents in ([64] : [63] > 0)

This answer will list all accident locations (field 64) where the number killed (field 63) is greater than zero.

If a filter is preceded by a pound sign (#), a count of matching items is returned, rather than the items themselves.

Question: how many fatal accidents has <Subject> been in

Answer: the driver has been in #([63] : [63] > 0) fatal accidents

This returns the number of items in field 63 where the value of the item in field 63 is greater than zero.

Conditions

Any particular answer statement will only be evaluated and output as a response if its associated condition is true. A condition is defined by using at least one comparison operator (=, !=, >, <, >=, and <= for numbers. eq, and ne for strings).

Question: how many points does <Subject> have

Condition1: [35] > 0

Answer1: the driver has [35] points

Condition2: [35] = 0

Answer2: the driver does not have any points

Conditions can include arithmetic operators, as in the example below.

Condition: [35] + [36] + [37] > 0

Compound conditions can be created by connecting simple conditions with AND and OR (& and |) operators.

Condition: [35] > [28] * 3 & [15] != 0 & [8] ne NONE

The above condition is true if the following three things are all true:

The value contained in field 35 is greater than three times the value in field 28.

The value in field 15 is not zero

The string in field 8 is not "NONE"

These rules also apply to the condition used in the second half of a filter.

Files

As mentioned above, the editor application can be used to create the grammar file needed by the SRE. This grammar file also contains information that is not used by the SRE, but is used directly by the runtime application. This includes the scripted answers, conditions, and weights in the case of a component system. This information is placed in comments that are ignored by the SRE. The runtime application reads this file at startup.

The runtime application also requires a file containing the record data to be used in the responses. The record data must be stored in a file called "record.txt", which has a specific format. Each line of the file contains one piece or record data, enclosed in brackets, preceded by two numbers in brackets. The first number is the field number for the field. This must match the number used when entering the sample questions and answers in the editor application. The second number is zero, unless the field has multiple values, in which case it specifies an index (starting from zero) to associate with the value.

[35][0][7]	< Points
[36][0][5]	< Last year's points
[37][0][Bob]	< Alias first name
[37][1][Frank]	<
[37][2][Stan]	<

CHAPTER VII

SAMPLE QUESTION SET OPTIMIZATION

For the purposes of this research, a spoken question answering system was developed. The domain of the system was driver records. The objective of the research was to compare different techniques for matching spoken user queries to a reasonable set of sample questions as might be used in a real application. The goals did not include comparing performance with less good sample questions to performance with better sample questions. Thus it was considered appropriate to refine the sample question set before collecting the final data for analysis.

The system was optimized using several cycles of data collection and analysis. This section describes the procedure used in the analysis of collected data and modification of the system based on that analysis. Modification of the system, for the most part, entailed reworking the set of sample questions used by the system. It also included fixing software bugs when discovered, as well as making changes in the way the data were collected. The modification was done in phases. Each phase represents the collection of data, an analysis of the data, and modifications made based on the analysis.

Phase 0 was an initial rough-draft phase. A question set was created using educated guesses about the queries subjects might pose. A group of subjects tested the system, and the results were analyzed, and shared with the

research committee. Using data gathered in phase 0, a new question set was developed.

In Phase 1, a new group of subjects tested the phase 1 question set, and the results were analyzed. The analysis shows an overall improvement of the system. The template set was then modified using phase 1 data.

The phase 2 question set was tested using the queries captured in phase 1. The analysis showed further improvement. The phase 2 question set was then tested using a group of new subjects. The results showed improvement over the phase 1 testing.

Phase 0

Once the software had been developed, a set of templates was required. This set of questions was created by making educated guesses concerning the queries that subjects might pose. The system was designed with a large amount of flexibility, allowing complex question forms to be represented. Thus, a rough draft set of questions included a number of such complex sample questions as well as simpler sample questions. It quickly became apparent that many of the more complex sample questions were not likely to be asked, and were impeding the quality of the voice recognition. The following questions are examples. Does the driver have more than 5 tickets for speeding in excess of 25 miles per hour? How many more points does the driver have for the current year than for last year?

While these are valid queries, and can be handled by the system, they are not likely to be asked. Given the state of voice recognition technology, it is preferable to include only questions that are likely to be asked, and not those that are asked very rarely or not at all. For this reason, the more complex sample questions were removed from the set. The result is the set of questions used for Test 0 shown below in figure 12.

Several rules are used, as shown by angle brackets(<>). For example, the <Subject> rule will match “the driver”, “he”, “she”, or “the person”.

Figure 3. Question Set 0

- 0 what is <2 PossessiveSubject> address
- 1 where does <3 Subject> live
- 2 what is <4 PossessiveSubject> <dob>
- 3 what is <5 PossessiveSubject> social security number
- 4 what type of license does <0 Subject> have
- 5 does <1 Subject> have any restrictions
- 6 does <Subject> have a valid license
- 7 what is the status of <6 PossessiveSubject> license
- 8 what is <0 PossessiveSubject> name
- 9 what does <1 Subject> look like
- 10 does <7 Subject> have any <aliases>
- 11 how many points does <9 Subject> have
- 12 does <9 Subject> have any points
- 13 does <10 Subject> have any convictions
- 14 has <10 Subject> ever been convicted
- 15 has the driver been convicted in the last <12 SingleDigit> years
- 16 does <14 Subject> have any speeding tickets
- 17 does <16 Subject> have any <osconv>
- 18 has <17 Subject> had any accidents
- 19 where has <18 Subject> had accidents
- 20 has <19 Subject> had any fatal accidents
- 21 how many accidents has <2 Subject> had
- 22 why was <3 PossessiveSubject> license suspended
- 23 does <15 Subject> have any D U I s
- 24 has <8 PossessiveSubject> license ever been suspended or revoked

A test involving subjects was used to exercise the template set. A summary of the analysis for the data collected is given below. For each system or weighting method tested during the optimization process, the number of proper responses is given. The percentage of proper responses with respect to the number of queries with reasonable templates is given in parentheses.

Phase 0 Analysis

Total sample questions with reasonable templates: 109

Of those 109,

Full question recognition responded properly to 30 (28%)

Component recognition (linear) responded properly to 38 (35%)

Component recognition (SIDF) responded properly to 35 (32%)

Component recognition (IDF) responded properly to 42 (39%)

Modifications

The results of this test were shared with the research committee. Changes to be made to the system were discussed. Questions that were not asked at all were removed. Missing questions that were asked were added to the set. New forms of questions that were asked were added to the set. Filler models were added where appropriate. In addition, a new rule (<Whats> = "whats" or "what is") was added. This resulted in the new set of sample questions used for phase 1 as shown in figure 13.

Figure 4. Question Set 1

0 <5 Whats> <0 PossessiveSubject> name
1 <5 Whats> <1 PossessiveSubject> last name
2 <0 Whats> <2 PossessiveSubject> first name
3 <5 Whats> <2 PossessiveSubject> address
4 where does <3 Subject> live
5 <5 Whats> the gender of <6 Subject>
6 what sex is <5 Subject>
7 <5 Whats> <4 PossessiveSubject> <dob>
8 when was <7 Subject> born
9 how tall is <8 Subject>
10 <5 Whats> <9 PossessiveSubject> height
11 how much does <10 Subject> weigh
12 <5 Whats> <11 PossessiveSubject> weight
13 what color is <12 PossessiveSubject> hair
14 <5 Whats> <13 PossessiveSubject> hair color
15 what color are <14 PossessiveSubject> eyes
16 <5 Whats> <15 PossessiveSubject> eye color
17 <5 Whats> <5 PossessiveSubject> social security number
18 what type of license does <0 Subject> have
19 does <1 Subject> have ... restrictions
20 does <1 Subject> have a valid license
21 <5 Whats> the status of <6 PossessiveSubject> license
22 has <8 PossessiveSubject> license ever been suspended or revoked
23 does <7 Subject> have ... <aliases>
24 how many points does <9 Subject> have
25 does <9 Subject> have ... points
26 does <10 Subject> have ... convictions
27 has <10 Subject> ever been convicted
28 how many convictions does <17 Subject> have
29 when was <18 Subject> convicted
30 what ... conviction dates
31 what types of convictions does <19 Subject> have
32 what has <20 Subject> been convicted for
33 does <22 Subject> have ... dee wees
34 does <0 Subject> have ... speeding tickets
35 has <17 Subject> had ... accidents
36 where has <18 Subject> had accidents
37 how many accidents has <2 Subject> had
38 what was the location of ... accidents

Phase 1

The system was again tested on new users. Data were collected to fine tune the system. The analysis of the data collected included the following.

1. The total number of queries
2. The number of queries that correspond to an answer in a sample question. These are referred to as reasonable template questions.
3. The number of queries that had sample questions that matched exactly.
4. The number of times a correct response was given using a full question recognition grammar.
5. The number of times a correct response was given using a component recognition grammar. For each question, three weighting methods were used (linear, SIDF, and IDF). The analysis includes a count of correct responses for each weighting method.
6. A table showing all responses given to all queries. In this table, each query is referred to as a record.
7. A list of likely causes for failure where one was apparent. The list includes a brief description of the problem, as well as the number of times it occurred. These are discussed in more detail following the analysis report given below.
8. A list of all spoken queries that were asked as transcribed from the wave files recorded during data collection. The list also specifies the number of times each query was asked.

Analysis of Test 1

There were a total of 77 records. Of these, 71 had reasonable templates and 42 had exactly matching templates.

Full question recognition responded properly 28 times.

Component recognition (linear) responded properly 24 times.

Component recognition (SIDF) responded properly 36 times.

Component recognition (IDF) responded properly 26 times.

The analysis record includes a table showing the sample questions chosen by each of the four methods, as well as the reasonable template if one exists. An excerpt of this table is shown below in table 3. The full table is given in Appendix A.

Table 2. Test 1 Summary

#	Wave File	Template	Full	Linear	SIDF	IDF
1	VictorAudio1-0.wav	0	13	13	13	13
2	VictorAudio1-3.wav	5	5	5	5	5
3	VictorAudio1-4.wav	7	21	21	7	21
4	VictorAudio1-5.wav	37	37	37	5	37
5	VictorAudio1-6.wav	12	12	5	12	5
6	VictorAudio2-5.wav	11	7	38	11	38

Spoken Queries in Order of Frequency:

During the analysis individual spoken queries were logged and counted.

The frequency of the individual queries is shown in table 4 below.

Table 4. Frequency of Spoken Queries

<u>Times Asked</u>	<u>Question</u>
7	what is the drivers address
6	what is the drivers date of birth
6	what is the drivers name
6	what is the drivers eye color
4	how many accidents has the driver been in
3	what are the drivers aliases
3	what is the drivers social security number
3	what is the drivers height
2	does the driver have any aliases
2	what is the drivers license number
2	how old is the driver
2	what is the drivers gender
2	does the driver have any convictions
2	what is the license status
2	what is the drivers license status
2	what type of convictions does the driver have
2	what is the drivers weight
1	what is the persons name
1	what is the gender of the driver
1	what is the weight of the driver
1	how much does this driver weigh
1	what color is the drivers eyes
1	does the driver have any license restrictions
1	how many points does the driver have
1	what is the drivers first name
1	what state was the drivers license issued in
1	what is the license type
1	what is the conviction type
1	what is the drivers conviction number
1	what is the drivers current license status
1	does the driver have any restrictions
1	what type of convictions
1	when were the drivers last convictions
1	how many convictions does the driver have
1	what are the dates of the drivers convictions
1	what color are the drivers eyes
1	what is the name of the driver
1	whats the name of the driver

Causes of error

Most of the errors that occurred during this test fall into one of the 5 categories listed below.

1. The use of rules that incorporate terms found in isolation. Some of the rules used in the templates include words that are also found in other questions.

An example is: `_0 <5 Whats> <0 PossessiveSubject> name`

The rule `<whats>` is found and given a term weight. Often, it is the case that the spoken input is, "what is the drivers name". Since the sample question

`13 what color is <12 PossessiveSubject> hair`

also contains the words "what" and "is", they are given more weight

independently. This additional weight overcomes the weight of the word "name" in the question. This results in a false response.

The solution is to not use rules that include words that occur independently in templates. A rule that contains synonyms is acceptable, as long as they do not occur where the rule is not used. The refined set of sample questions does not use the rules `<Whats>` and `<dob>` as they have been found to result in incorrect responses.

2. Unused templates. Once again, it is apparent that questions that are not asked only serve to degrade system performance. It is desirable that the question set includes questions that are likely to be asked often, and not questions that are rarely asked, or not asked at all.

The solution is to use the statistics gathered in the Test 1 analysis to determine which questions are asked frequently and which are not in a

quantitative manner. Questions that are not likely to be asked should be removed from the set.

3. Questions not represented by templates. In some cases the system failed because there was no sample question that could result in a proper response. For the most part, these are requests for information that the system is not intended to give, or oddly worded queries. Examples are, “why were you pulled over”, and “excuse me, what is your name”.

The solution is to add any missing questions that have been asked multiple times. Given observation 2 above, it is better to omit rarely asked questions. At this point, most questions that have been asked do have a reasonable template, so only minor modifications were made to address this issue, and only if tests showed that a question is likely to be asked somewhat frequently.

4. Speech recognition error caused by quiet input. Although subjects were asked to speak loudly, clearly, and directly into the microphone, some of the subjects did not. In some cases, a subject would sit back in the chair and talk far too quietly. In other cases, the subject spoke clearly and directly into the microphone. In the latter cases, the same types of errors are not found. The only reasonable solution is to be more demanding when asking participants to speak up.

5. General speech recognition errors. There are times when the speech recognition fails due to a subject’s intonation, accent, or other vocal artifacts. There is no solution to this problem.

Modification

Using the data collected in Test 1, it was determined that some pieces of information are requested more often than others, while some pieces of information are not requested at all. In addition, certain question phrasings were shown to be common.

Below is a list of the most common pieces of information requested, and a grammar phrasing that matches the actual phrasing used. For each piece of information, the total number of times requested (out of 77 inquiries) is given. Each grammar rule is preceded by the number of times a particular phrasing matched the rule given. The list includes a total of 13 sample questions, which represent 45 of the queries actually asked (58%).

Full name – 9 times

7 what is <PossessiveSubject> name

2 what is the name of <Subject>

Eye color – 8 times

6 what is <PossessiveSubject> eye color

2 what color are <PossessiveSubject> eyes

Address – 7 times

7 what is <PossessiveSubject> address

Date of birth – 6 times

6 what is <PossessiveSubject> date of birth

Aliases – 5 times

3 what are <PossessiveSubject> aliases

2 does <Subject> have ... aliases

Number of accidents – 4 times

4 how many accidents has <Subject> been in

Weight – 4 times

2 what is <PossessiveSubject> weight

1 how much does <Subject> weigh

1 what is the weight of <Subject>

License status – 5 times

2 what is <PossessiveSubject> license status

Given the frequency of these questions, they were included in the new set of sample questions to be used for Test 2. In addition to the questions represented above, a number of questions were asked with lower frequency. These templates were also be included in the new set of templates. Each sample question below is preceded by the number of times it was asked.

- 3 what is <PossessiveSubject> height
- 3 what is <PossessiveSubject> social security number
- 2 does subject have ... restrictions
- 2 does <Subject> have ... convictions
- 2 what type of convictions does <Subject> have
- 2 what is <PossessiveSubject> gender
- 1 what is the gender of <Subject>
- 1 how many convictions does <Subject> have
- 1 how many points does <Subject> have
- 1 what is <PossessiveSubject> first name
- 1 what is <PossessiveSubject> license type

The addition of these 19 templates makes a set of 24 templates that accounts for 64 of the 77 questions asked (83%).

Finally, it was noted that while the driver's eye color was asked for 8 times, no subjects inquired about the driver's hair color. Looking at the Driver Record Tree the subjects were given, this is the one piece of information that was not asked for. It can be assumed that future test subjects may request this information, so the following templates were added, based on the phrasing of the similar eye color templates.

- what is <PossessiveSubject> hair color
- what color is <PossessiveSubject> hair

The new sample question set has a total of 26 templates. Of these, 17 appeared in the former set, which had a total of 39 templates. Thus 22 questions, which were shown to be ineffective, were removed, and 9 new questions were added. The questions for template set 2 are shown in Figure 14.

Figure 5. Sample Question Set 2

- 0 what is <2 PossessiveSubject> first name
- 1 what is <2 PossessiveSubject> address
- 2 what is the gender of <6 Subject>
- 3 what is <1 PossessiveSubject> gender
- 4 what is <4 PossessiveSubject> date of birth
- 5 what is <9 PossessiveSubject> height
- 6 how much does <10 Subject> weigh
- 7 what is <11 PossessiveSubject> weight
- 8 what is the weight of <2 Subject>
- 9 what color is <12 PossessiveSubject> hair
- 10 what is <13 PossessiveSubject> hair color
- 11 what color are <14 PossessiveSubject> eyes
- 12 what is <15 PossessiveSubject> eye color
- 13 what is <5 PossessiveSubject> social security number
- 14 what is <0 PossessiveSubject> license type
- 15 does <1 Subject> have ... restrictions
- 16 what is <0 PossessiveSubject> license status
- 17 what is <0 PossessiveSubject> name
- 18 what is the name of <3 Subject>
- 19 does <7 Subject> have ... <aliases>
- 20 what are <4 PossessiveSubject> aliases
- 21 how many points does <9 Subject> have
- 22 what type of convictions does <19 Subject> have
- 23 does <10 Subject> have ... convictions
- 24 how many convictions does <5 Subject> have
- 25 how many accidents has <2 Subject> been in

Verifying the Modification

Using the audio files gathered during Test 1, the phase 2 question set was tested to show that its modifications led to improvement with respect to the phase 1 question set it. The results were analyzed as Test 1b. The analysis from Test 1 is also shown for comparison. The percentages listed below are with reference to the number of queries with reasonable templates.

Test 1

Full question recognition responded properly 39%

Component recognition (linear) responded properly 33%

Component recognition (SIDF) responded properly 50%

Component recognition (IDF) responded properly 36%

Test1b

Full question recognition responded properly 68%

Component recognition (linear) responded properly 65%

Component recognition (SIDF) responded properly 63%

Component recognition (IDF) responded properly 67%

The phase 2 question set shows a dramatic improvement over the phase 1 set when used with the phase 1 query data.

Phase 2

The next step was to show that the new question set performed well with new queries. The phase 2 question set was tested using a new group of subjects. The Analysis shows that the correct response rate has improved for new queries.

Test 2

Full question recognition responded properly 60%

Component recognition (linear) responded properly 72%

Component recognition (SIDF) responded properly 75%

Component recognition (IDF) responded properly 72%

Summary

Table 4 below shows the correct response ratio for all methods of matching, and for all phases.

Table 4. Optimization Test Summary

Phase	0	1	1b	2
Correct responses to all queries				
Full question recognition	19%	36%	61%	44%
Component (linear)	24%	31%	61%	53%
Component (SIDF)	22%	46%	57%	55%
Component (IDF)	27%	33%	60%	53%
Correct responses to queries with reasonable templates				
Full question recognition	28%	39%	68%	60%
Component (linear)	35%	33%	65%	72%
Component (SIDF)	32%	50%	63%	75%
Component (IDF)	39%	36%	67%	72%

Given that the correct response ratio is significantly improved, and all methods respond successfully greater than 50% of the time (all systems succeeded more often than not), it was decided that the data collected in phase 2 are valid for the purposes of analysis. The remaining data for this research were collected in the same manner.

CHAPTER VIII

DATA COLLECTION

Prior to data collection, the Institutional Review Board at UNH Research Conduct and Compliance Services was contacted. They provided a release form to be signed by each subject under IRB number 2980. A copy of this form is included as Appendix C. The signed forms were faxed to the IRB for tracking.

Test subjects were isolated in a room during their questioning. The testing area consisted of a chair, and a desk. A computer and microphone were positioned on the desk. Each subject was instructed to sit in the chair facing the computer. The subjects were asked to speak loudly and clearly, and directly into the microphone. They were also instructed to ask a number of questions of their choosing. The test subjects were provided with two documents to explain the test. The first, figure 15, is an instruction sheet titled Ask Fred. This sheet explains the context of the test, and provides instructions. The second, figure 16, is a tree diagram depicting the types of data contained within the record. These documents are shown on the next two pages.

Each time a subject asked a question, Fred responded, and stored an audio copy of the question as a wave file. A total of 417 questions were asked. These audio files were then processed and analyzed as described in the next section.

CHAPTER IX

ANALYSIS

Processing

Data Blocks

The 417 audio files collected were processed by a full system and a component system. A data block as shown below was created for each query.

Figure 17. Data Block

Test Question: 8			
Wave File: FredAudio6-8.wav			
Spoken Query: what is his eye color			
Reasonable Sample Question:M 12 13 14			
Full SR Response: what is his eye color			
Full Selected Question: 14, 14#what is <15 PossessiveSubject> eye color			
Full System Response: the driver has brown eyes			
Comp SR Response: what is eye color			
Comp Selected Question: 14, 14#what is <15 PossessiveSubject> eye color			
Comp System Response: the driver has brown eyes			
Component System Candidates:			
Weighting Scheme	Linear	SIDF	IDF
First Choice (Score)	14 (256)	14 (141)	14 (60)
Second Choice (Score)	11 (156)	11 (37)	11 (43)
Third Choice (Score)	12 (156)	12 (37)	12 (43)
Fourth Choice (Score)	13 (116)	13 (31)	13 (29)
Fifth Choice (Score)	0 (68)	0 (11)	0 (28)

Each line of the data block is explained below.

Test Question: This is a batch number that was only useful during processing.

Wave File: This is the name of the audio file containing the query.

Spoken Query: This is the spoken query as transcribed from the wave file.

Reasonable Sample question: A reasonable sample question is an acceptable match according to the guidelines discussed below. A query may have multiple reasonable sample questions. This is also the test used to determine if a spoken query is reasonable. If a query has one or more reasonable sample questions, it is a reasonable query. If the query is worded exactly the same as any sample question, the reasonable sample question(s) is preceded by an "M".

Full SR Response: This is the string of text returned by the SR of the full system. Since it is a full system, the string will be identical to one of the sample questions unless the SR could not find an acceptable string, in which case a question mark (?) is returned.

Full Selected Question: This is the sample question number chosen by the full system.

Full System Response: This is the response given from the full system as an answer to the user's query.

Comp SR Response: This is the string of text returned by the SR of the component system. Since it is a component system, the string will not necessarily be identical to any of the sample questions.

Comp Selected Question: This is the sample question number chosen by the component system based on the string returned from the SR.

Comp System Response: This is the response given from the component system.

Component System Candidates: The lower section of the data block is a table showing the top five choices for three different weighting schemes. Linear refers to the linear weighting, SIDF refers to the literal inverse document frequency, and IDF refers to the commonly used IDF function involving the log of the inverse document frequency. For each weighting measure, the five highest ranking sample questions are given, along with the calculated scores. The binary weighting method was not included until a later stage of processing.

Reasonable Sample Questions

A reasonable sample is a sample question that will provide an answer to the query posed. During processing, if no reasonable sample question existed

for a query, a "-1" was entered. Choosing reasonable questions is somewhat subjective. The criteria are listed below.

A spoken question was considered to have a reasonable sample question if:

- The query asked for information that was contained in the record.
- A sample question existed that returned the requested information.
- The key words in the query were contained in the SR grammar file.

Where the key words are the domain specific words that normally refer to a piece of information, such as points, address, or convictions.

The following queries would not have reasonable sample questions.

Is the driver married?

Does he require spectacles?

Information concerning a person's marital status is not included in the driver record. Although the record does contain restrictions, including the requirement for corrective lenses, since the word "spectacles" is not in the grammar there is no reason the system would legitimately choose a sample question that would result in an acceptable answer.

The data blocks are saved as TestData.txt. A summary of this information is given in Appendix A.

Analysis

Of the 417 query files, 268 had reasonable sample questions. The queries without reasonable sample questions were removed from the analysis and are not discussed further.

Four different weighting methods were used in the analyses that follow. These weighting methods are discussed in detail in Chapter 4, and are summarized here.

Linear – A linear function giving a value of 100 to very rare words (appearing in only one sample question), and a value of 0 to very common words (appearing in all sample questions).

IDF – The traditional log of the inverse document frequency. In this case, the natural logarithm is used.

SIDF – The simple IDF; the literal inverse document frequency function without taking the logarithm.

Binary – Each word has a weight of 1.

To compare these weighting methods, a new table was generated that indicates whether or not each of the four methods succeeded in returning an appropriate response for each of the 268 reasonable questions. Only the first candidate is used for linear, IDF, and SIDF weighting. For the binary weighting method, the audio files were reprocessed and only the top score was considered. This table is included in Appendix A.

Comparison of Systems and Weighting Methods

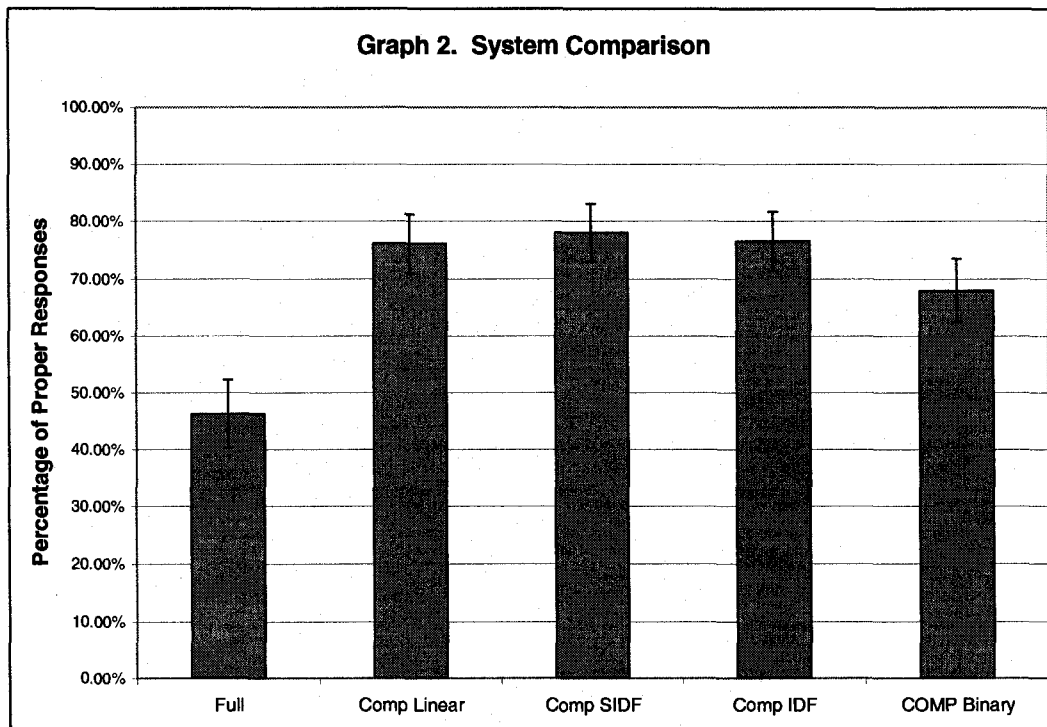
For the main comparison, all 268 data blocks were processed by both a full system and a component system. In addition, the component system applied four weighting methods for comparison.

The table below compares the success of the implementations, by listing the number of queries each implementation responded to properly out of the total 268. All results of proportion (percentage correct) are shown along with the corresponding confidence intervals computed at the 95% confidence level, using the conventional method based on the normal distribution (Ross, 2003).

Table 4. Comparison of Systems and Weighting Methods.

System	Proper Responses	Percentage of Total
Full	124	46.3% \pm 6.0%
Component Linear	204	76.1% \pm 5.1%
Component SIDF	209	78.0% \pm 5.0%
Component IDF	205	76.5% \pm 5.1%
Component Binary	182	67.9% \pm 5.6%

As the table shows, the component system was most successful, particularly when the system used varying weights (non-binary). Given that the top three systems (Linear, IDF, SIDF) were all within 2% of each other (which was within the margin or error), no significant difference in performance was detected between these component systems. All three of these implementations performed significantly better than the full system.



Given that the three best component implementations did not result in significant differences in performance for these data, the remaining analysis will focus on a comparison of the full system and the component system with linear weighting. Note that it would be necessary to collect and analyze at least 6000 spoken queries in order to reduce the confidence interval to +/- 1% in order to test the possible significance of the differences seen between the three techniques. This was not feasible in the current research.

Impact of Speech Recognition

It was expected that the SR performance would suffer in the component system due to smaller grammar items (single words versus multi-word sample

questions), and more grammar items (“#X” single words versus “#Y” multi-word sample questions). As a measure of the SR performance in the full system, the percentage of proper responses to exact matches was calculated. This represents the number of inputs the SR correctly matched, given the pool of items the recognizer was expected to match.

For comparison, as a measure of SR performance in the component system, the percentage of correctly recognized words was calculated. Again, this represents the number of inputs the SR matched, given the pool of items the recognizer was expected to match.

Table 7. Comparison of Speech Recognition Performance

System	Total	Recognized	% Recognized
Full (recognized matches)	83	75	90.4% ± 6.4%
Component (recognized words)	1055	735	69.7% ± 2.8%

We can see that for "fair" inputs, the component system has inferior SR performance.

Another way to measure the impact of the SR performance on the component system is by using "perfect recognition". To simulate perfect SR, the component system was run using the transcribed questions (Spoken Query) for all 268 data blocks. The results of the linear weighted system are compared to those of the same system using actual SR.

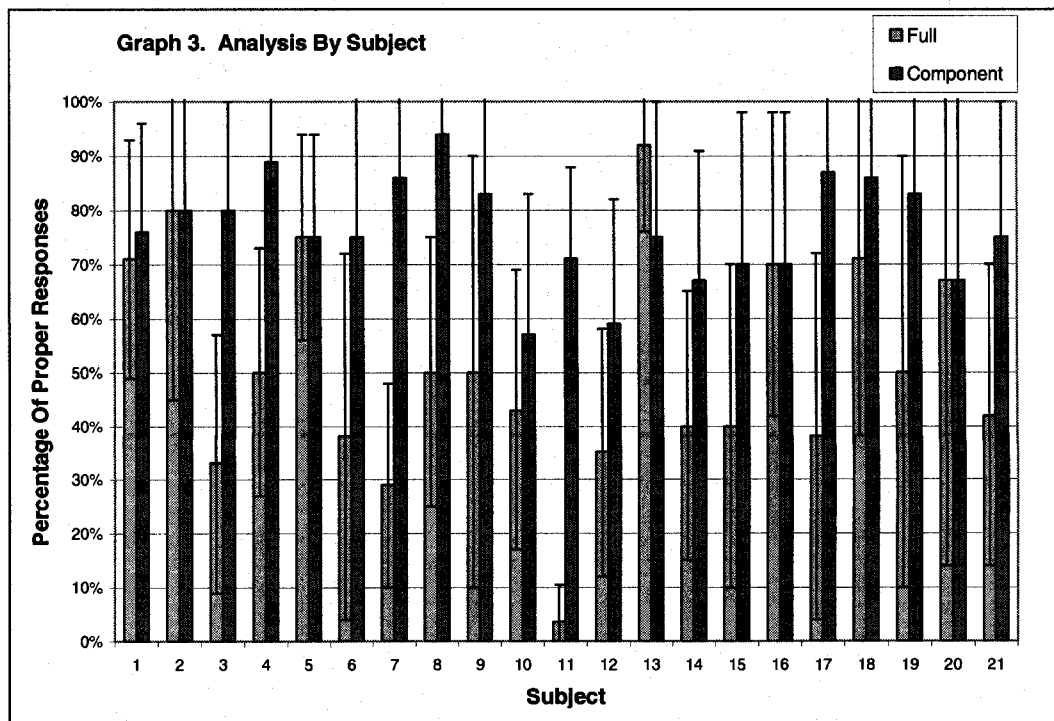
Table 6. Actual Versus Transcribed

Component System Linear Weighting	Proper Responses	Percentage of Total
Transcribed (Perfect SR)	220	82.1% ± 4.6%
Actual SR	204	76.1% ± 5.1%

While the system with perfect SR appears to perform better, these results are within the margin of error, and not conclusive.

Analysis by Subject

An analysis by test subject shows that the systems responded differently to different subjects. Graph 3 below shows the percentage of queries the full system and component system (linear only) responded to properly for each subject. Given the limited number of questions recorded from each subject, the resulting margins of error are large, but some general trends can be identified.



The graph shows that the component system performed significantly better than the full system for 5 of the 21 subjects (24%). For the other subjects, no statistically significant difference can be reported, although the trend seems to lean towards the component system. There was only a single case in which the computed performance of the full system exceeded that of the component system. The main factor contributing to the difference appears to be the way in which subjects phrased their queries.

Subject queries can be divided into two categories; anticipated, and not anticipated. Some of the queries were phrased exactly as anticipated (they matched a sample question).

What is the drivers name?

What is the drivers eye color?

What is the drivers date of birth?

Does the driver have any aliases?

Queries phrased as anticipated usually resulted in a proper response from both systems. In addition, queries that were phrased very closely to a sample question often resulted in a proper response from both systems.

Some queries were phrased considerably differently than anticipated. Subjects 3, 4, 7, 8, and 11 used unanticipated phrasing frequently, often consisting of single keywords. The examples below do not match any sample question, and were responded to properly by the component system, but not by the full system.

What are the restrictions?

Points?

What gender is he?

Aliases?

Date of birth?

Type of license?

Are there any accidents on the drivers record?

How many points is on the license?

How many accidents has he had before?

type of license?

any restrictions?

eye color?

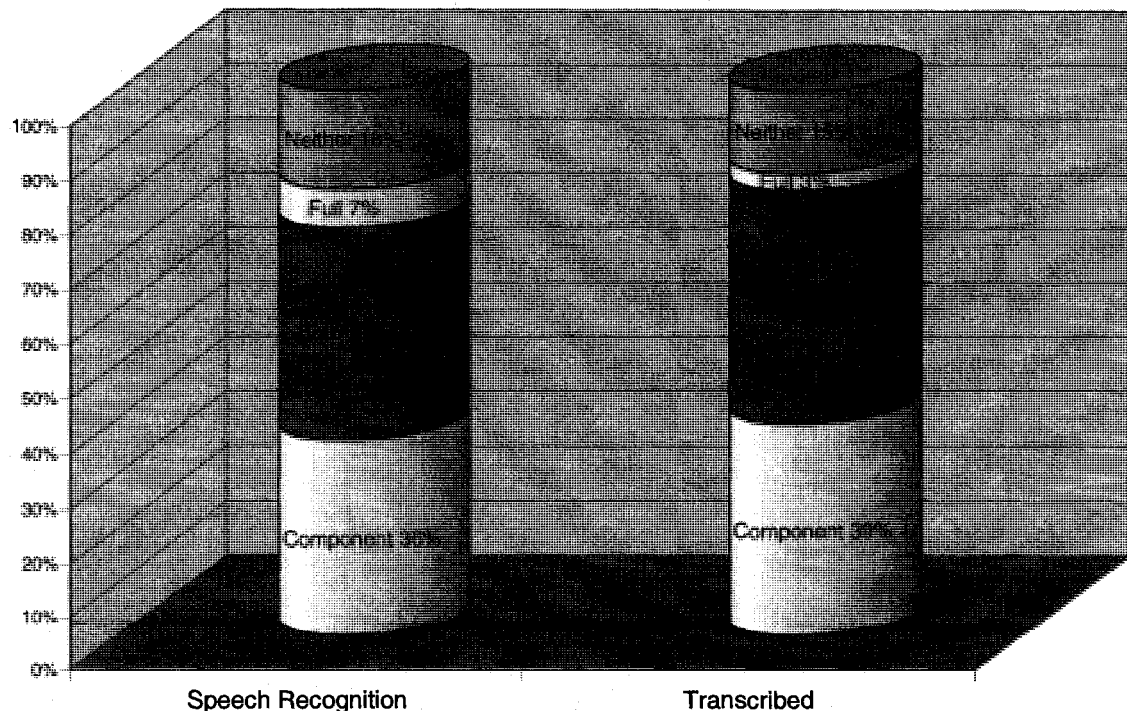
birthday?

Overlap

It is worth noting the query overlap between the two systems. As shown on the left in graph 2 below, 39% of the queries asked were responded to appropriately by both systems. However, there were a number of questions (7%) that the full system correctly responded to, and the component system did not. The component system succeeded on 36% of the questions that the full system failed on. These questions were, for the most part, not phrased as anticipated. The remaining 18% of the questions were not responded to properly by either system.

When transcribed queries were used to simulate perfect SR, as shown on the right, the increase in correct responses for the component system was drawn to the "Component" and "Both" categories from the "Neither" and "Full" categories. That is, the component system responded properly to some questions that it failed on with actual SR, but the full system succeeded on. The component also responded properly to some questions that neither system succeeded on previously. Note that in the "Transcribed" graph on the right, transcriptions were used only for the component system. The full system used actual SR in both cases.

Graph 4. Overlap in System Success Using Both Speech Recognition and Transcriptions



CHAPTER X

CONCLUSIONS

A spoken question answering system that uses full question recognition is likely to succeed most of the time when the question asked is identical to a sample question. A system that uses component word recognition has the potential to respond to additional questions, but is more likely to make speech recognition errors as it recognizes words individually rather than in full sentences. The purpose of this study was to examine how the benefit of the flexibility offered by the component recognition compares to the loss in speech recognition performance.

While other studies have explored the use of cosine similarity scores to compare short text strings, past research has not addressed the issue of the impact of speech recognition as it applies to such systems, or to closed-domain SQA systems in general. It has been suggested that recognition errors are compensated for when the target document is large. However, the impact of recognition errors on short queries has not been explored to the extent that it has in this study.

In systems that apply cosine similarity scores, classical tf-idf weighting is always used with very little variation. No attempt has been made in past studies to examine weighting schemes other than tf-idf when used in similar applications where short text strings are compared, and specifically in domain-specific SQA systems. This study examined the difference between several weighting

methods, and examined the impact of speech recognition errors on such systems by both comparing the performance to a so called "full" system, and comparing the performance to a system with "perfect" speech recognition by the use of query transcripts.

The system described uses a set of template questions to which each user query is compared using several variations of the cosine similarity measure with tf-idf weighting. These variations were compared to see if any showed a significant performance benefit. No indexing techniques, such as stemming, synonym expansion, morphological expansion, n-gram featuring, or stop lists were employed, although the functionality afforded by some of these techniques is embedded in the ability to use grammar rules. The SRE used a grammar containing only words that appear in the template questions, rather than the common large vocabulary ASR.

Results of Analysis

Considering all participants, the component system (with linear weighting) responded properly to about 76% of the questions, while the full question system responded properly to only about 46% of the questions. This difference corresponds to the advantage gained in using this component word recognition system over the full system.

As expected, the component system made frequent recognition errors, and only recognized about 69% of the words properly. Using transcribed questions, the component system responded properly to about 82% of the

questions. This shows that a substantial increase in recognition errors (31%), might result in only a small decrease of overall system success (6%).

Considering only the queries that were identical to sample questions, the full system responded properly to about 90% of the questions, while the component system responded properly to about 86%. This shows that for predictable questions, the loss in speech recognition puts the component system at a disadvantage. The full system's 90% recognition rate corresponds to the component system's 69% recognition for individual words. We can see that the component system does make recognition errors more frequently.

The component word recognition system assigns weights to each word. For this study, four variations of the tf-idf weighting commonly used in Internet search engines were used in parallel. The weighting schemes included a standard implementation of the common IDF function, a less linear SIDF function (using the raw inverse document frequency), a linear function, and a binary weight (0 or 1). The different weighting methods place more or less importance on word rarity. No significant difference was observed between weighting schemes in the analysis, although the results suggest that the binary weighting may be less effective than the other three, although it also performed significantly better than the full system.

In an analysis by subject, the component system performed significantly better than the full system for about 24% of the participants. These subjects tended to phrase queries in unanticipated ways, and often used short phrases or

single keywords. There were no cases in which the full system significantly outperformed the component system.

In the past, researchers have shied away from using component word recognition in spoken question answering systems because of the negative impact on speech recognition performance. Studies have shown that when the text being recognized is long, such as in a spoken document, individual recognition errors are compensated for by the redundancy and context contained within the text. However, this claim is not valid when the recognized text is a much shorter query. It has been acknowledged that for query recognition, a single recognition error could have a profound impact, and that recognition errors are an issue for any language based technology that recognizes small spans of text (Allen, 2002).

In open-domain SQA research, it has been found that query recognition errors cause a substantial performance loss as compared to the same system using transcribed inputs (Schofield, 2003). No studies have examined the impact of speech recognition in closed-domain systems by comparing the success of the system with recognized and transcribed inputs. This study shows that with the reduced grammar size and sample question set inherent in a closed-domain system, recognition errors have a much smaller effect on system success as compared to open-domain SQA systems.

Current QA and SQA systems that use cosine similarity scores implement the standard tf-idf weighting method almost without exception. Most current QA systems are document retrieval systems. It has been shown that this similarity

score for document retrieval does not work well when relatively shorter questions are the targets (Jeon, 2005). No attempt has been made to examine the appropriateness of this weighting scheme in closed-domain SQA systems which have very short target "documents". Other weighting methods have not been directly compared to tf-idf weighting in such systems. This study compared three different weighting methods to the traditional IDF function, and did not discover a significant difference between them in this application, although the results suggest that a measure of rarity (as opposed to binary weighting), offers useful information for the comparison.

Software Developed

The development system has many features that allow developers to create domain-specific spoken question answering systems. The editor application is used to design the system. It offers a graphical representation of the query structure that provides system organization. Sample questions and answers can be placed in a logical structure, and are written in a simple scripting language. The language supports grammar rules to increase question flexibility. Each question can be associated with multiple answers, which are chosen at run time based on conditional statements. The conditions and answers may include counts and comparisons of data items, and the scripting language has support for basic arithmetic and Boolean functions.

Once the questions and answers have been defined, the editor application creates all the files required by the runtime application, with the exception of the

data record. The editor application can generate files to create a full system, or a component system. The runtime application automatically runs in the proper mode based on the files supplied by the editor application. Once started, the runtime application will continue to answer questions until closed.

Recommended Use

The SQA development system is designed to allow developers to design domain-specific spoken answering systems quickly and easily. Based on the experience gained in this study, the following steps are recommended.

1. Gather a group representative of the intended system users. Have them ask questions as if they were using the finished system, and record the exact phrasing of their questions.
2. Choose to build either a full question system, or a component word system. Based on an analysis of the questions asked, one type of system may be preferable for the application. A full question system might be the best choice if the intended users will be trained, or will be using the system many times, or if there are only a small number of predictable questions to which the system will need to respond.
3. Build a question/answer set based on the questions asked in step one. Refine the set as necessary.

For component word systems, it is not necessarily helpful to have multiple phrasings of a question, such as:

What is the student's grade point average?

How high is the student's grade point average?

What does the student have for a grade point average?

While useful in a full question system, multiple phrasings in the component system dilute the effectiveness of the key words in the question.

An answer such as, "yes", is not as helpful as, "yes, the student is passing". Include feedback in the answer, so the user is alerted if the system has misunderstood the question.

For either type of system, including more sample questions will allow the system to respond to more inputs, but is also likely to result in more recognition errors. Include commonly asked questions, and ones that are necessary for the system to have. Do not include oddly worded questions, or questions that are very rarely asked.

Looking Forward

There are several improvements that could be made to this system. As it is, the data file must be parsed into a specific format before the runtime application can answer questions. This means that a parser must be written for each question answering system. It would be convenient if the runtime

application could read a SQL query result in a standard format, such as XML or CSV. The editor application could also read this SQL query result, and build the tree structure based on the query metadata, automatically linking the tree structure to the data items in the runtime application.

An important result of this study is that the SQA system made a significant number of errors in selecting the appropriate sample question even when perfect speech recognition was simulated by using human transcriptions in place of the speech recognizer output. Thus, improvements in speech recognizer performance alone may not be sufficient to make SQA systems of the type studied useful. Further research is needed both with the aim of improving the original selection of the set of sample questions and with the aim of improving the scoring algorithm used to select the best member of a set of sample questions in response to a specific spoken query.

Any SQA system must have a means to represent acceptable queries of some form. While this form has received much attention, less has been paid to which queries are best to represent. As seen in this experiment, sample questions with common words interfere in positive and negative ways. A study concerning the relationships between sample questions in similar systems would be beneficial to SQA system design.

Another approach to choosing sample questions would be to use user feedback to modify the sample question list. The main challenge here would be in the addition of new questions containing words not currently in the lexicon.

Given advances in speech recognition performance a user-flagged misrecognized sentence might be sent to an ASR module to discern new words.

The results of this research indicate that the SQA system performance was not highly sensitive to the fixed word weights used for computing matching scores, as long as the weights used place more emphasis on less commonly occurring words. Thus, further research specifically aimed at improving the approach to defining fixed word weights may not be fruitful, unless those weights consider some other factor in addition to frequency of occurrence. Some speech recognition software has the ability to provide confidence scores for the choices made, and to provide alternative choices for each spoken word also tagged with relative confidence scores. Further research is needed to determine how best to incorporate these word confidence scores and alternative choices for a spoken query into the weighting scheme for the component word recognition, along with the fixed weights based on frequency of occurrence in the sample question set.

LIST OF REFERENCES

- Aho, A. (1968). Indexed grammars - an extension of context-free grammars. *Journal of the ACM*, 15(4), 647-671.
- Akiba, T., Fujii, A., and Itou, K. (2004). Effects of language modeling on speech-driven question answering. *Proceedings of the 8th International Conference on Spoken Language Processing (ICSLP 2004)*, Jeju Island, Korea, October 4-8. 1053-1056.
- Albrecht, D.W., Zukerman, I., Nicholson, A.E., and Bud, A. (1997). Towards a Bayesian model for keyhole plan recognition in large domains. *Proceedings of the Sixth International Conference on User Modeling*, Sardinia, Italy, June 2-5. 365-376.
- Allen, J. F., Miller, B. W., Ringger, E. K., and Sikorski, T. (1996). A robust system for natural language dialog. *Proceedings of the 1996 Annual Meeting of the Association for Computational Linguistics (ACL '96)*, Santa Cruz, California, June 1996. 62-70.
- Allan, J (2002) Perspectives on information retrieval and speech. *ACM SIGIR Forum Archive* 36(1). 1-10.
- Barnett, J., Anderson, S., Broglio, J., Singh, M., Hudson, R., and Kuo, S. W. (1997). Experiments in spoken queries for document retrieval. *Proceedings of Eurospeech97*, Rhodes, Greece, September 22-25. 1323-1326.
- Bilotti, M., Ogilvie, P., Callan, J., Nyberg, E. (2007). Structured retrieval for question answering. *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. Amsterdam, July 23-27. 351-358.
- Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd, T. (1977). GUS, A frame-driven dialogue system. *Artificial Intelligence*. 8(2), 155-173.
- Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*. 21(4), 543-566.
- Burke, R., Hammond, K., Kulyukin, V., Lytinen, S., Tomuro, N., Schoenberg, S. (1997). Natural language processing in the FAQ Finder system: Results and prospects. *AAAI Spring Symposium on Natural Language Processing for the World Wide Web*, Palo Alto, California, March 24-25. 17-26.

- Carde, C., Ng, V., Pierce, D., and Buckley, C. (2000). Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering system. *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP-2000)*, Seattle, Washington, April 29 - May 4. 180-187.
- Chomsky, N. (1956). Three models for the description of language. *IRI Transactions on Information Theory*. 2(3), 113-124.
- Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, 2, 137-167.
- Crestani, F. (2002). Spoken query processing for interactive information retrieval. *Data & Knowledge Engineering*, 41. 105-124.
- Demner-Fushman, D., and Lin, J. (2005). Knowledge extraction for clinical question answering: preliminary results. *Workshop on question answering in restricted domains. 20th National Conference on Artificial Intelligence (AAAI-05)*, Pittsburgh, Pennsylvania, July 9-13. 1-9.
- Dumias, S., Banko, M., Brill E., Lin, J., Ng, A. (2002). Web question answering: is more always better? *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, Tampere, Finland, August 11-15. 291-298.
- Dunning, T. (1993) Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1), pp 61-74, 1993.
- Francis, W. N. (1979). A tagged corpus – Problems and prospects. In Greenbaum, S., Leech, G., and Svartvik, J. (Eds), *Studies in English linguistics for Randolph Quirk*. (pp. 192-209). London and New York: Longman.
- Franz, A., and Milch, B. (2002). Searching the web by voice. *Proceedings of the Conference on Computational Linguistics (COLING)*, Taipei, Taiwan, August 24 to September 1. 1213-1217.
- Fujii, A., Itou, K. (2004). An evaluation of the Web retrieval task at the third NTCIR workshop. *ACM SIGIR Forum Archive*, 38(1). 39-45.
- Galitsky, B. (2002). A tool for extension and restructuring natural language question answering domains. *15th IEA/AIE Conference*. Cairns, Australia. June 17-20. 282-292.

- Garside, R. (1987). The CLAWS word tagging system. In Garside, R., Leech, G., and Sampson, G. (Eds) *The Computational Analysis of English*. London and New York: Longman.
- Garside, R., Leech, G., and McEnery, A. (1997). *Corpus Annotation*. London and New York: Longman.
- Godfrey, J., Holliman, E., and McDaniel, J. (1992). SWITCHBOARD: Telephone speech corpus for research and development. In *IEEE ICASSP-92*. 517-520.
- Goto, J., Miyazaki, M., Kobayakawa, T., Hiruma, N., Uratani, N. (2006). A TV agent system that integrates knowledge and answers users' questions. *IUI '06: Proceedings of the 11th international conference on Intelligent user interfaces*, Sydney, Australia, January 29 to February 1. 300-302.
- Greene, B. B., and Rubin, G. M. (1971). *Automatic grammatical tagging of English*. Providence, RI: Brown University.
- Gupta, N., Rahim, M., and Riccardi, G. (2007). System for handling frequently asked questions in a natural dialog service. U.S. Patent 7197460.
- Halliday, M. A. K. (1978). *Language as Social Semiotic: The social interpretation of language and meaning*. London: Edward Arnold.
- Harabagiu, H., Moldovan, D., Picone, J. (2002). Open-domain voice-activated question answering. *Proceedings of the 19th International Conference on Computational Linguistics*, Taipei, Taiwan, August 24 to September 1. 1-7.
- Harris, Z. S. (1962). *String analysis of sentence structure*. Mouton, The Hague.
- Hidaka, N., Masui, F. (2003). A comparison of answer ranking methods in question answering. *Proceedings of the Annual Conference of JSAI*, 17(2) 1-3.
- Hedstrom, A. (2005). *Question categorization for a question answering system using a vector space model*. Master's thesis, Department of Linguistics and Philology (Language Technology Programme) Uppsala University, Uppsala, Sweden.
- Hemphill, C. T., Godfrey, J., and Doddington, G. R. (1990). The ATIS spoken language systems pilot corpus. In *Proceedings DARPA Speech and Natural Language Workshop*. Hidden Valey, PA. 96-101. Morgan Kaufmann.

- Hewitt, C. (1971). Description and Theoretical Analysis of Planner. PhD Thesis. Massachusetts Institute of Technology.
- Hofstadter, D. R. (1979). Gödel, Escher, Bach: An eternal golden braid. New York: Basic Books.
- Horvitz, E., and Paek, T. (2001). Harnessing models of users' goals to mediate clarification dialog in spoken language systems. In Proceedings of the Eighth Conference on User Modeling. Sonthofen, Germany, July 13-17. 3-13.
- Hovy, E. H., Gerber, L., Hermjakob, U., Lin, C., Ravichandran, D. (2001). Towards semantics-based answer pinpointing. Proceedings of the DARPA Human Language Technology Conference. San Diego, CA, March 18-21. 1-7.
- Jeon, J., Croft, W. B., Lee, J.H. (2005). Finding semantically similar questions based on their answers. SIGIR '05: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Salvador, Brazil, August 15 to 19. 617-618.
- Jijkoun, V. and Rijke, M. (2005). Retrieving answers from frequently asked questions pages on the web. Proceedings of the 14th ACM international conference on Information and knowledge management. Bremen, Germany, October 31 to November 15. 76-83.
- Johnson, R. (1983). Parsing with transition networks. In King, M. (Ed) Parsing natural language. 59-72. London: Academic Press.
- Joshi, A. K. (1985). Tree adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions? In Dowty, D. R., Karttunen, L., and Zwicky, A. (Eds.), Natural language parsing, 206-250. Cambridge: Cambridge University Press.
- Jurafsky, D., and Martin, J. H. (2000). Speech and Language Processing. New Jersey: Prentice Hall.
- Kaplan, R., and Bresnan, J. (1982). Lexical functional grammar – A formal system for grammatical representation. In Bresnan, J. (Ed.), The mental representation of grammatical relations, (pp. 173-281). Cambridge, MA: MIT Press.
- Katz, B. (1997). From sentence processing to information access on the World Wide Web. AAAI Spring Symposium on Natural Language Processing for the World Wide Web, Palo Alto, California, March 24-25. 77-94.

- Katz, B., Lin, J., Felshin, S. (2002) The START multimedia information system: current technology and future directions. In Proceedings of the International Workshop on Multimedia Information Systems, Tempe, Arizona, October 10 to November 1. 117-123.
- Kay, P., and Fillmore, C. J. (1999). Grammatical constructions and linguistic generalizations. The What's X doing Y? construction. *Language*, 75(1), 1-33.
- Klein, S., and Simmons, R. F. (1963). A computational approach to the grammatical coding of English words. *Journal of the Association for Computing Machinery*, 10(3), 334-347.
- Kučera, H., and Francis, W. N. (1967). *Computational analysis of present day American English*, Providence, RI: Brown University Press.
- Lau, R. (1997). WEBGALAXY: Beyond point and click – A conversational interface to a browser. Proceedings Sixth International World Wide Web Conference, Santa Clara, CA, April 7-11. 119-127.
- Leuski, A., Patel, R., and Traum, D. (2006). Building effective question answering characters. Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue. Sydney, Australia, July 15-16. 18-27.
- Lin, J. (2007). An exploration of the principles underlying redundancy-based factoid question answering. *ACM Transactions on Information Systems (TOIS)*, 25(2), Article 6, 1-55.
- Luhn, H. P. (1957). A statistical approach to the mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4), 309-317.
- Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2), 313-330.
- Marshall, I. (1983). Choice of grammatical word-class without global syntactic analysis: Tagging words in the LOB corpus. *Computers and the Humanities*, 17, 139-150.
- Mlynarczyk, S., and Lytinen, S. (2005). Faqfinder question answering improvements using question/answer matching. In Proceedings of L&T-2005 - Human Language Technologies as a Challenge for Computer Science and Linguistics, Poznan, Poland, April 21-23.

- Molla, D., Vicedo, J. (2007) Question Answering in Restricted domains: an overview. *Computational Linguistics*, 30(1), 41-61.
- Narayanan, S. and Harabagiu, S. (2004). Question answering based on semantic structures. *Proceedings of the 20th international conference on Computational Linguistics*, Geneva, Switzerland, August 23-27.
- Otterbacher, J., Radev, D. (2004). Comparing semantically related sentences: the case of paraphrase versus subsumption. *Proceedings of the 20th International Conference on Computational Linguistics*, August 23-27. Article No. 1265.
- Hidaka, N., and Fumito, M. (2003). A comparison of answer ranking methods in question answering. *Proceedings of the Annual Conference of JSAI*, 17(2), 1-3.
- Padó, s., Lapata, M. (2007). Dependency-Based Construction of Semantic Space Models. *Computational Linguistics*, 33(2). 161-199.
- Paşca, M. (2007). Lightweight web-based fact repositories for textual question answering. *CIKM '07: Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management*, Lisboa, Portugal, November 6-9. 87-96.
- Peters, Stanley, and Ritchie (1982). *Phrase linking grammars*. Technical Report. Department of Linguistics, University of Texas at Austin.
- Prager, J., Brown, E., Coden, A., and Radeu, D. (2000). Question – answering by predictive annotation. In *Proceedings 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Athens, Greece, July 24-28. 184-191.
- Radev, D., Fan, W., Qi, H., Wu, H., Grewal. A. (2002). Probabilistic Question Answering on the Web. *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, May 7-11. 408-419.
- Ross, T. D. (2003). Accurate confidence intervals for binomial proportion and Poisson rate estimation. *Computers in Biology and Medicine* 33, 509-531.
- Roussinov, D., and Robles-Flores, J. (2004). Self-learning web question answering system. *Proceedings of the 13th International World Wide Web Conference*, New York City, NY, May 17-22. 400, 401.
- Salton, G. (1971). *The SMART retrieval system: Experiments in automatic document processing*. Englewood Cliffs, NJ: Prentice Hall.

- Schofield, E. and Zheng, Z. (2003). A speech interface for open-domain question-answering. Proceedings of the 41st Annual Meeting on Association for Computational Linguistics, 2(2), 177-180.
- Seneff, S., and Polifroni, J. (1996). A new restaurant guide conversational system: Issues in rapid prototyping for specialized domains. Proceedings International Conference on Spoken Language Processing, Philadelphia, PA. October 3-6. 665-668.
- Shannon, C. E. (1948). A mathematical theory of communication. Bell System Technical Journal, 27(3), 379-423.
- Shinyama, Y., Sekine, S., Sudo, K., Grishman, R. (2002). Automatic paraphrase acquisition from news articles. To appear in the Proceedings of Human Language Technology Conference (HLT 2002), San Diego, CA, March 24-27. 313-318.
- Sneiders, E. (2002). Automated question answering using question templates that cover the conceptual model of the database. Proceedings of the 6th International Conference on Applications of Natural Language to Information Systems-Revised Papers. Stockholm, Sweden, June 27-28. 235-239.
- Soricut, R., Brill, E. (2004). Automatic question answering: beyond the factoid. Proceedings of the Human Language Technology and North American Association for Computational Linguistics Conference (HLT/NAACL-2004). May 2-5, Boston, MA.
- Soricut, R., and Brill, E. (2006). Automatic question answering: beyond the factoid. Information Retrieval, 9(2), 191-206.
- Spark Jones, K. (1972). A statistical interpretation of term frequency and its application in retrieval. Journal of Documentation, 28(1), 11-21.
- Stolz, W. S., Tannenbaum, P. H., and Carstensen, F. V. (1965). A stochastic approach to the grammatical coding of English. Communications of the ACM, 8(6), 399-405.
- Tanenbaum, A. S. (1992). Modern Operating Systems. Prentice Hall, NJ.
- Tsur, O., de Rijke, M., and Sima'an, K. (2004). BioGrapher: Biography questions as a restricted domain question answering task. In Workshop on Question Answering in Restricted Domains. 42nd Annual Meeting of the Association for Computational Linguistics (ACL-2004), Barcelona, Spain, July 21-26. 23-30.

- Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society, 42, 230-265. Read to the society in 1936, but published in 1937.
- Turing, A. M. (1950). Computing machinery and intelligence. Mind, 59, 433-460.
- Weizenbaum, J. (1966). ELIZA – A computer program for the study of natural language communication between man and machine. Communications of the ACM, 9(1), 36-45.
- Wiegand, M., Leidner, J., Klakow, D. (2007). Combining term-based and event-based matching for question answering. Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Amsterdam, July 23-27. 715-716.
- Winograd, T. (1972). Understanding natural language. Cognitive psychology, 3(1).
- Woods, W. A. (1970). Transition network grammars for natural language analysis. Communications of the ACM, 13, 591-606.
- Wu, C., Yeh, J., Chen, M. (2005). Domain-specific FAQ retrieval using independent aspects. ACM Transactions on Asian Language Information Processing (TALIP), 4(1), 1-17.
- Wu, Y., Hu, X., Kashioka, H. (2007) Mining redundancy in candidate-bearing snippets to improve web question answering. Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, Lisboa, Portugal, November 6-9. 999-1002.
- Wundt, W. (1900). Völkerpsychologie: eine untersuchung der entwicklungsgesetze von sprache, und Sitte, W. Engelmann, Leipzig. Band II: Die Sprache, Zweiter Teil.
- Zue, V. (1995). Navigating the information superhighway using spoken language interfaces. IEEE Expert 10, 39-43.

APPENDIX A

TEST DATA

Test 1

Test 1 was a preliminary test used to optimize the system. The table below shows the data for the 77 questions gathered. For each question, the table contains the wave file name, the reasonable template, and the templates chosen by the full question recognition system, as well as the component recognition system result for all three weighting methods. The wave file name is used as a unique identifier. If no reasonable template question exists, a "-1" was entered.

Table 7. Test 1 Data Summary

Wave File	Template	Full	Linear	SIDF	IDF
VictorAudio1-0.wav	0	13	13	13	13
VictorAudio1-3.wav	5	5	5	5	5
VictorAudio1-4.wav	7	21	21	7	21
VictorAudio1-5.wav	37	37	37	5	37
VictorAudio1-6.wav	12	12	5	12	5
VictorAudio2-5.wav	11	7	38	11	38
VictorAudio2-6.wav	15	0	8	8	8
VictorAudio2-11.wav	19	33	21	19	21
VictorAudio2-12.wav	24	24	24	24	24
VictorAudio2-13.wav	23	23	23	23	23
VictorAudio2-14.wav	2	2	2	2	2
VictorAudio2-15.wav	-1	0	21	17	21
VictorAudio2-16.wav	-1	12	21	17	21
VictorAudio2-17.wav	-1	12	21	21	21
VictorAudio2-18.wav	3	3	38	3	38

Wave File	Template	Full	Linear	SIDF	IDF
VictorAudio2-19.wav	3	3	3	3	3
VictorAudio2-20.wav	-1	0	9	9	9
VictorAudio2-21.wav	-1	9	5	5	5
VictorAudio2-23.wav	7	21	38	7	38
VictorAudio3-0.wav	0	0	0	0	0
VictorAudio3-1.wav	7	0	33	33	33
VictorAudio3-2.wav	16	10	38	38	38
VictorAudio3-3.wav	23	12	21	8	21
VictorAudio3-5.wav	16	0	16	16	16
VictorAudio3-6.wav	17	0	21	21	21
VictorAudio3-7.wav	18	0	18	18	18
VictorAudio3-8.wav	5	0	5	5	5
VictorAudio3-9.wav	7	0	21	21	21
VictorAudio3-10.wav	31	30	8	8	8
VictorAudio3-11.wav	-1	13	13	30	13
VictorAudio3-13.wav	23	13	13	13	13
VictorAudio3-14.wav	10	0	10	10	10
VictorAudio3-16.wav	3	3	13	13	13
VictorAudio3-17.wav	3	0	3	3	3
VictorAudio3-18.wav	7	34	38	34	38
VictorAudio4-0.wav	26	0	5	5	5
VictorAudio4-1.wav	0	13	21	21	21
VictorAudio4-2.wav	0	0	38	38	38
VictorAudio4-3.wav	17	0	38	17	38
VictorAudio4-5.wav	21	12	8	8	8
VictorAudio4-6.wav	21	0	21	38	21
VictorAudio4-7.wav	10	10	38	15	38
VictorAudio4-8.wav	37	0	35	35	35
VictorAudio4-9.wav	23	23	23	23	23
VictorAudio4-10.wav	0	13	38	38	38
VictorAudio4-11.wav	3	3	38	3	38
VictorAudio4-12.wav	3	3	38	3	38
VictorAudio4-13.wav	37	0	23	23	23
VictorAudio4-14.wav	16	0	38	4	38
VictorAudio4-15.wav	16	16	38	38	38
VictorAudio4-16.wav	21	12	21	21	21
VictorAudio4-17.wav	7	21	38	7	38
VictorAudio4-18.wav	19	19	19	19	19
VictorAudio4-19.wav	28	34	28	28	28
VictorAudio4-20.wav	31	0	18	18	18
VictorAudio4-21.wav	31	18	38	38	38
VictorAudio4-22.wav	0	13	38	38	38
VictorAudio4-23.wav	0	13	13	13	13

Wave File	Template	Full	Linear	SIDF	IDF
VictorAudio4-24.wav	5	3	5	5	5
VictorAudio4-25.wav	3	3	38	3	38
VictorAudio4-26.wav	17	17	17	17	17
VictorAudio4-27.wav	10	10	13	18	13
VictorAudio4-28.wav	12	12	13	12	12
VictorAudio4-29.wav	15	13	15	15	15
VictorAudio4-30.wav	29	29	1	1	1
VictorAudio4-31.wav	37	37	37	37	37
VictorAudio4-32.wav	28	28	28	28	28
VictorAudio4-33.wav	29	21	38	30	38
VictorAudio4-34.wav	31	31	26	38	26
VictorAudio4-35.wav	21	3	21	21	21
VictorAudio5-8.wav	12	12	13	12	12
VictorAudio5-9.wav	16	13	13	18	13
VictorAudio5-10.wav	15	15	15	15	15
VictorAudio5-11.wav	0	5	5	5	5
VictorAudio5-12.wav	21	12	38	33	38
VictorAudio5-13.wav	23	32	38	23	38
VictorAudio5-17.wav	0	0	5	5	5

Final Test Data

The final test data report is similar to the Test 1 data report above. For each question, the table contains a number (#) used in batch processing, the wave file name, the reasonable template, and the templates chosen by the full question recognition system, as well as the component recognition system result for all three initial weighting methods. The wave file name is used as a unique identifier. If no reasonable template question exists, "None" was entered.

Table 8. Final Test Data

#	Wave File	Template	Full	Linear	SIDF	IDF
0	FredAudio6-0.wav	0	0	19	19	19
1	FredAudio6-1.wav	20	0	20	20	20
2	FredAudio6-2.wav	3	3	3	3	3
3	FredAudio6-3.wav	6	6	6	6	6
4	FredAudio6-4.wav	7	7	7	7	7
5	FredAudio6-5.wav	15	15	15	15	15
6	FredAudio6-6.wav	9	9	9	9	9
7	FredAudio6-7.wav	11	11	11	19	11
8	FredAudio6-8.wav	14	14	14	14	14
9	FredAudio6-9.wav	19	19	19	19	19
10	FredAudio6-10.wav	22 23	22	23	23	23
11	FredAudio6-11.wav	25	19	21	25	21
12	FredAudio6-12.wav	25	19	21	25	21
13	FredAudio6-15.wav	17	0	17	17	17
14	FredAudio6-16.wav	None	19	21	14	21
15	FredAudio6-20.wav	None	19	14	14	14
16	FredAudio6-21.wav	None	19	6	6	6
17	FredAudio6-22.wav	0	0	0	0	0
18	FredAudio6-23.wav	None	19	19	19	19
19	FredAudio6-24.wav	22 23	22	23	23	23
20	FredAudio6-25.wav	None	0	15	15	15
21	FredAudio6-28.wav	25	19	21	25	21
22	FredAudio7-0.wav	19	19	19	19	19
23	FredAudio7-1.wav	0 1	0	1	1	1
24	FredAudio7-2.wav	None	19	24	1	24
25	FredAudio7-3.wav	18	18	18	18	18
26	FredAudio7-4.wav	None	19	19	19	19
27	FredAudio7-5.wav	None	22	15	15	15
28	FredAudio7-6.wav	None	22	21	18	21
29	FredAudio7-7.wav	None	0	1	1	1
30	FredAudio7-8.wav	4	0	4	4	4
31	FredAudio7-9.wav	None	1	1	1	1
32	FredAudio7-10.wav	None	0	1	1	1
33	FredAudio7-11.wav	None	0	1	1	1
34	FredAudio7-12.wav	None	19	1	16	1
35	FredAudio7-13.wav	None	0	10	10	10
36	FredAudio7-14.wav	None	3	1	20	1
37	FredAudio7-15.wav	3	3	1	3	1
38	FredAudio7-16.wav	None	0	15	15	15
39	FredAudio8-0.wav	0	0	0	0	0
40	FredAudio8-1.wav	15	0	15	15	15

#	Wave File	Template	Full	Linear	SIDF	IDF
41	FredAudio8-2.wav	21	0	21	21	21
42	FredAudio8-5.wav	11 12	11	11	6	11
43	FredAudio8-6.wav	None	20	1	1	1
44	FredAudio8-7.wav	25	22	25	25	25
45	FredAudio8-8.wav	25	0	25	25	25
46	FredAudio8-10.wav	16	24	24	1	24
47	FredAudio8-11.wav	10	10	10	25	10
48	FredAudio8-12.wav	8 9	7	8	8	8
49	FredAudio8-13.wav	18	0	1	18	1
50	FredAudio8-14.wav	15	15	15	15	15
51	FredAudio8-17.wav	None	0	25	25	25
52	FredAudio8-18.wav	25	0	25	25	25
53	FredAudio8-21.wav	7	0	3	3	3
54	FredAudio8-22.wav	7	7	7	7	7
55	FredAudio8-23.wav	16	0	16	16	16
56	FredAudio9-0.wav	0 1	0	1	3	1
57	FredAudio9-1.wav	0 1	9	15	15	15
58	FredAudio9-2.wav	None	0	8	8	8
59	FredAudio9-3.wav	3	3	3	3	3
60	FredAudio9-4.wav	6	0	6	6	6
61	FredAudio9-5.wav	None	0	8	8	8
62	FredAudio9-6.wav	None	0	19	19	19
63	FredAudio9-7.wav	7	0	7	7	7
64	FredAudio9-8.wav	None	0	8	8	8
65	FredAudio9-9.wav	22	22	22	18	22
66	FredAudio9-10.wav	24	24	24	24	24
67	FredAudio9-11.wav	25	0	25	25	25
68	FredAudio9-12.wav	25	23	25	25	25
69	FredAudio9-13.wav	3	3	3	3	3
70	FredAudio9-14.wav	None	0	15	15	15
71	FredAudio9-15.wav	15	0	15	15	15
72	FredAudio9-16.wav	11	11	11	11	11
73	FredAudio9-17.wav	13	0	13	13	13
74	FredAudio9-18.wav	13	0	16	8	16
75	FredAudio9-19.wav	17	17	17	17	17
76	FredAudio9-20.wav	18	18	18	18	18
77	FredAudio9-21.wav	16	16	16	15	16
78	FredAudio9-22.wav	20	5	20	20	20
79	FredAudio10-0.wav	1	1	1	1	1
80	FredAudio10-1.wav	4	0	4	4	4
81	FredAudio10-2.wav	4	4	4	4	4
82	FredAudio10-3.wav	22	22	1	1	1
83	FredAudio10-4.wav	24	0	14	14	14

#	Wave File	Template	Full	Linear	SIDF	IDF
84	FredAudio10-5.wav	25	19	25	25	25
85	FredAudio10-6.wav	25	19	21	25	21
86	FredAudio10-7.wav	25	0	25	25	25
87	FredAudio10-8.wav	21	21	1	21	1
88	FredAudio10-9.wav	6	6	6	6	6
89	FredAudio10-10.wav	25	25	1	19	1
90	FredAudio10-11.wav	24	24	24	24	24
91	FredAudio10-12.wav	18	18	18	18	18
92	FredAudio10-13.wav	17	17	17	17	17
93	FredAudio10-14.wav	6	6	6	6	6
94	FredAudio10-15.wav	19	19	19	19	19
95	FredAudio10-16.wav	7	7	7	7	7
96	FredAudio10-17.wav	10	10	10	10	10
97	FredAudio10-18.wav	None	10	1	8	1
98	FredAudio10-19.wav	3	3	3	3	3
99	FredAudio10-20.wav	13 14	14	13	13	13
0	VictorAudio14-0.wav	15	5	2	2	2
1	VictorAudio14-1.wav	15	15	15	15	15
2	VictorAudio14-2.wav	13 14	0	11	11	8
3	VictorAudio14-4.wav	None	0	15	15	15
4	VictorAudio14-5.wav	None	0	1	1	1
5	VictorAudio14-6.wav	None	0	1	1	1
6	VictorAudio14-7.wav	7	7	7	7	7
7	VictorAudio14-8.wav	None	0	16	16	16
8	VictorAudio14-9.wav	21	0	21	21	21
9	VictorAudio14-10.wav	25	25	25	25	25
10	VictorAudio14-11.wav	None	0	25	25	25
11	VictorAudio14-12.wav	6	0	6	6	6
12	VictorAudio14-13.wav	4	0	4	4	4
13	VictorAudio14-14.wav	None	0	16	16	16
14	VictorAudio15-0.wav	0	0	0	0	0
15	VictorAudio15-1.wav	14	14	14	14	14
16	VictorAudio15-2.wav	None	0	16	16	16
17	VictorAudio15-3.wav	None	0	19	19	19
18	VictorAudio15-4.wav	None	0	19	19	19
19	VictorAudio15-5.wav	None	0	1	1	1
20	VictorAudio15-6.wav	21	0	1	21	1
21	VictorAudio15-7.wav	None	10	8	8	8
22	VictorAudio15-8.wav	3	3	3	3	3
23	VictorAudio15-9.wav	4	4	4	4	4
24	VictorAudio15-10.wav	15	5	15	15	15
25	VictorAudio15-11.wav	None	0	1	1	1
26	VictorAudio15-12.wav	22	0	22	22	22

#	Wave File	Template	Full	Linear	SIDF	IDF
27	VictorAudio15-13.wav	24	20	13	13	13
28	VictorAudio15-14.wav	18	0	18	18	18
29	VictorAudio15-15.wav	17	0	17	17	17
30	VictorAudio15-16.wav	16	7	16	16	16
31	VictorAudio15-17.wav	6	0	6	6	6
32	VictorAudio15-18.wav	7	0	7	7	7
33	VictorAudio15-19.wav	11	0	11	11	11
34	VictorAudio15-20.wav	11	11	11	11	11
35	VictorAudio15-21.wav	8	8	8	8	8
36	VictorAudio15-22.wav	9	0	9	9	9
37	VictorAudio15-23.wav	None	0	19	19	19
38	VictorAudio15-24.wav	None	0	19	19	19
39	VictorAudio15-25.wav	24	20	13	13	13
40	VictorAudio15-26.wav	25	0	25	25	25
41	VictorAudio15-27.wav	19	0	19	19	19
42	VictorAudio15-28.wav	21	0	21	21	21
43	VictorAudio15-29.wav	None	0	0	0	0
44	VictorAudio15-30.wav	None	6	1	3	3
45	VictorAudio15-31.wav	None	1	1	3	3
46	VictorAudio16-0.wav	0	0	0	13	0
47	VictorAudio16-1.wav	3	0	0	0	0
48	VictorAudio16-2.wav	3	11	3	3	3
49	VictorAudio16-3.wav	5 4	5	4	4	4
50	VictorAudio16-4.wav	6	6	6	6	6
51	VictorAudio16-5.wav	15	15	15	15	15
52	VictorAudio16-6.wav	7	7	7	7	7
53	VictorAudio16-7.wav	9	9	9	9	9
54	VictorAudio16-8.wav	11	11	11	25	11
55	VictorAudio16-9.wav	14	0	14	14	14
56	VictorAudio16-10.wav	18	5	18	18	18
57	VictorAudio16-11.wav	16	0	16	16	16
58	VictorAudio16-12.wav	17	0	17	17	17
59	VictorAudio16-13.wav	17	19	17	17	17
60	VictorAudio16-14.wav	19	19	19	19	19
61	VictorAudio16-15.wav	None	0	15	15	15
62	VictorAudio16-16.wav	25	0	25	25	25
63	VictorAudio16-18.wav	None	0	0	0	0
64	VictorAudio16-19.wav	None	19	1	19	19
65	VictorAudio16-20.wav	None	0	19	19	19
66	VictorAudio17-0.wav	0	0	0	13	0
67	VictorAudio17-1.wav	None	0	10	10	10
68	VictorAudio17-2.wav	19	0	19	19	19
69	VictorAudio17-3.wav	25	17	18	18	18

#	Wave File	Template	Full	Linear	SIDF	IDF
70	VictorAudio17-4.wav	25	0	25	25	25
71	VictorAudio17-5.wav	4 5	5	4	4	4
72	VictorAudio17-6.wav	None	8	13	13	13
73	VictorAudio17-7.wav	None	22	1	19	19
74	VictorAudio17-8.wav	None	0	0	0	0
75	VictorAudio17-9.wav	None	1	16	16	16
76	VictorAudio17-10.wav	3	3	3	3	3
77	VictorAudio18-0.wav	14	0	7	7	7
78	VictorAudio18-1.wav	8	0	8	8	8
79	VictorAudio18-7.wav	3	0	0	0	0
80	VictorAudio18-8.wav	3	0	3	3	3
81	VictorAudio18-9.wav	8	0	8	8	8
82	VictorAudio18-10.wav	0	0	19	19	19
83	VictorAudio18-11.wav	0 1	1	1	25	1
84	VictorAudio18-12.wav	23	0	11	22	11
85	VictorAudio18-13.wav	8	8	8	8	8
86	VictorAudio18-14.wav	4	4	4	4	4
87	VictorAudio18-15.wav	None	0	2	2	2
88	VictorAudio18-16.wav	None	3	17	17	17
89	VictorAudio18-17.wav	18	18	18	18	18
90	VictorAudio18-20.wav	None	0	15	15	15
91	VictorAudio18-21.wav	20	1	17	17	17
92	VictorAudio18-22.wav	20	0	0	0	0
93	VictorAudio18-23.wav	3	3	3	3	3
94	VictorAudio18-24.wav	None	0	2	2	2
95	VictorAudio18-25.wav	None	0	17	17	17
96	VictorAudio19-0.wav	0	0	0	0	0
97	VictorAudio19-1.wav	21	0	21	21	21
98	VictorAudio19-2.wav	22	0	22	22	22
99	VictorAudio19-3.wav	22	0	22	22	22
100	VictorAudio19-4.wav	25	0	25	25	25
101	VictorAudio19-5.wav	25	0	15	15	15
102	VictorAudio19-6.wav	25	0	15	15	15
103	VictorAudio19-7.wav	6	0	18	18	18
104	VictorAudio19-8.wav	6	0	6	6	6
105	VictorAudio19-9.wav	25	0	25	25	25
106	VictorAudio19-10.wav	25	0	15	15	15
107	VictorAudio19-11.wav	11	0	11	11	11
108	VictorAudio19-12.wav	18	0	18	18	18
109	VictorAudio19-15.wav	17	0	17	17	17
110	VictorAudio19-16.wav	14	0	7	7	7
111	VictorAudio19-17.wav	14	0	14	14	14
112	VictorAudio19-19.wav	9	0	6	6	6

#	Wave File	Template	Full	Linear	SIDF	IDF
113	VictorAudio19-20.wav	9	0	9	9	9
114	VictorAudio19-23.wav	None	0	0	0	0
115	VictorAudio19-24.wav	19	0	19	19	19
116	VictorAudio19-25.wav	None	0	2	2	2
117	VictorAudio19-26.wav	None	0	0	0	0
118	VictorAudio19-27.wav	6	0	6	6	6
119	VictorAudio19-28.wav	None	0	13	13	13
120	VictorAudio19-29.wav	None	0	16	16	16
121	VictorAudio19-30.wav	None	0	15	15	15
122	VictorAudio19-31.wav	None	0	1	1	1
123	VictorAudio19-32.wav	None	0	19	19	19
124	VictorAudio19-33.wav	17	0	17	17	17
125	VictorAudio19-35.wav	16	0	16	16	16
126	VictorAudio19-36.wav	None	0	14	14	14
127	VictorAudio19-37.wav	None	0	1	1	1
128	VictorAudio19-38.wav	24	0	7	7	7
129	VictorAudio19-39.wav	23	0	15	15	15
130	VictorAudio19-40.wav	None	0	13	13	13
131	VictorAudio19-41.wav	None	0	15	15	15
132	VictorAudio19-43.wav	None	0	16	16	16
133	VictorAudio19-46.wav	None	0	19	19	19
134	VictorAudio19-49.wav	None	0	25	25	25
135	VictorAudio19-50.wav	6	0	6	6	6
136	VictorAudio19-51.wav	None	0	4	4	4
137	VictorAudio19-52.wav	None	0	8	8	8
138	VictorAudio19-54.wav	None	0	11	11	11
139	VictorAudio19-55.wav	14	0	14	14	14
140	VictorAudio19-56.wav	11	0	11	11	11
141	VictorAudio19-57.wav	None	0	8	8	8
142	VictorAudio19-58.wav	None	0	15	15	15
143	VictorAudio19-59.wav	None	0	25	25	25
144	VictorAudio19-60.wav	None	0	0	0	0
145	VictorAudio19-61.wav	None	0	0	0	0
146	VictorAudio19-63.wav	None	0	13	13	13
147	VictorAudio19-65.wav	3	0	3	3	3
0	VictorAudio20-0.wav	0	0	0	0	0
1	VictorAudio20-1.wav	3	0	3	3	3
2	VictorAudio20-3.wav	4	0	4	4	4
3	VictorAudio20-4.wav	8	8	8	8	8
4	VictorAudio20-5.wav	None	0	0	0	0
5	VictorAudio20-6.wav	14	0	0	0	0
6	VictorAudio20-7.wav	18	0	18	18	18
7	VictorAudio20-8.wav	14	0	15	15	15

#	Wave File	Template	Full	Linear	SIDF	IDF
8	VictorAudio20-9.wav	14	14	16	16	16
9	VictorAudio20-10.wav	11	11	11	19	11
10	VictorAudio20-11.wav	14	14	14	14	14
11	VictorAudio20-12.wav	15	0	15	15	15
12	VictorAudio20-13.wav	6	5	15	15	15
13	VictorAudio20-14.wav	6	0	19	19	19
14	VictorAudio20-15.wav	6	0	6	6	6
15	VictorAudio20-16.wav	7	7	13	13	13
16	VictorAudio20-17.wav	7	0	13	13	13
17	VictorAudio20-18.wav	None	0	1	1	1
18	VictorAudio20-19.wav	19	0	19	19	19
19	VictorAudio21-0.wav	0	0	8	8	8
20	VictorAudio21-1.wav	0 1	1	1	1	1
21	VictorAudio21-2.wav	3	3	3	3	3
22	VictorAudio21-3.wav	4	4	4	4	4
23	VictorAudio21-4.wav	6	6	6	6	6
24	VictorAudio21-5.wav	15	15	15	15	15
25	VictorAudio21-6.wav	7	7	7	7	7
26	VictorAudio21-7.wav	10	10	10	10	10
27	VictorAudio21-8.wav	11 12	11	11	11	11
28	VictorAudio21-9.wav	14	14	14	14	14
29	VictorAudio21-10.wav	18	3	1	18	1
30	VictorAudio21-11.wav	17	17	1	17	1
31	VictorAudio21-12.wav	None	0	16	8	16
32	VictorAudio21-13.wav	None	19	8	8	8
33	VictorAudio22-0.wav	None	0	8	8	8
34	VictorAudio22-1.wav	None	11	15	15	15
35	VictorAudio22-2.wav	13	0	13	13	13
36	VictorAudio22-4.wav	19	19	18	18	18
37	VictorAudio22-5.wav	None	10	20	20	20
38	VictorAudio22-6.wav	22	22	22	22	22
39	VictorAudio22-7.wav	None	20	13	13	13
40	VictorAudio22-8.wav	None	0	1	25	1
41	VictorAudio22-9.wav	5	5	5	5	5
42	VictorAudio22-10.wav	None	0	4	4	4
43	VictorAudio22-11.wav	None	18	6	6	6
44	VictorAudio22-12.wav	None	0	16	16	16
45	VictorAudio22-13.wav	None	0	3	3	3
46	VictorAudio22-14.wav	8	8	8	8	8
47	VictorAudio22-15.wav	None	0	3	3	3
48	VictorAudio22-16.wav	None	8	3	3	3
49	VictorAudio22-17.wav	21	21	10	21	21
50	VictorAudio22-18.wav	None	0	25	25	25

#	Wave File	Template	Full	Linear	SIDF	IDF
51	VictorAudio22-19.wav	None	19	16	25	16
52	VictorAudio22-20.wav	None	4	15	15	15
53	VictorAudio22-21.wav	None	18	15	15	15
54	VictorAudio22-22.wav	None	8	25	25	25
55	VictorAudio22-23.wav	None	20	15	15	15
56	VictorAudio22-25.wav	25	0	15	15	15
57	VictorAudio22-26.wav	None	21	13	13	13
58	VictorAudio22-27.wav	None	0	25	25	25
59	VictorAudio22-28.wav	None	8	24	15	24
60	VictorAudio22-29.wav	None	0	0	0	0
61	VictorAudio22-30.wav	None	0	0	0	0
62	VictorAudio22-31.wav	None	0	8	8	8
63	VictorAudio22-32.wav	7	0	7	7	7
64	VictorAudio22-33.wav	None	0	15	15	15
65	VictorAudio22-34.wav	18	5	18	18	18
66	VictorAudio22-35.wav	17	0	17	17	17
67	VictorAudio22-36.wav	17	0	13	17	13
68	VictorAudio22-37.wav	17	5	17	17	17
69	VictorAudio22-38.wav	None	0	3	3	3
70	VictorAudio22-39.wav	None	0	15	15	15
71	VictorAudio22-40.wav	None	0	15	15	15
72	VictorAudio22-41.wav	None	4	6	6	6
73	VictorAudio22-42.wav	None	0	0	0	0
74	VictorAudio22-43.wav	None	0	16	16	16
75	VictorAudio22-47.wav	None	0	4	25	4
76	VictorAudio22-48.wav	None	0	8	8	8
77	VictorAudio22-49.wav	19	0	19	19	19
78	VictorAudio22-50.wav	None	0	18	18	18
79	VictorAudio22-51.wav	None	0	25	25	25
80	VictorAudio22-52.wav	22	0	22	22	22
81	VictorAudio22-53.wav	None	20	13	13	13
82	VictorAudio22-54.wav	None	0	22	22	11
83	VictorAudio22-55.wav	5	5	15	15	15
84	VictorAudio22-56.wav	None	0	8	8	8
85	VictorAudio22-57.wav	None	0	0	0	0
86	VictorAudio23-0.wav	0	0	0	0	0
87	VictorAudio23-1.wav	15	0	15	15	15
88	VictorAudio23-2.wav	22 23	22	23	23	23
89	VictorAudio23-3.wav	None	20	13	8	13
90	VictorAudio23-4.wav	None	0	8	8	8
91	VictorAudio23-5.wav	11 12	11	11	11	11
92	VictorAudio23-6.wav	25	0	15	15	15
93	VictorAudio23-7.wav	25	0	25	25	25

#	Wave File	Template	Full	Linear	SIDF	IDF
94	VictorAudio23-8.wav	19 20	19	20	20	20
95	VictorAudio23-9.wav	19	10	19	19	19
96	VictorAudio23-10.wav	3	0	1	1	1
97	VictorAudio23-11.wav	3	4	1	3	1
98	VictorAudio24-0.wav	0	0	11	11	11
99	VictorAudio24-1.wav	3	3	3	3	3
100	VictorAudio24-2.wav	0 1	1	1	1	1
101	VictorAudio24-3.wav	11 12	11	11	11	11
102	VictorAudio24-4.wav	4	4	4	4	4
103	VictorAudio24-5.wav	None	3	1	20	1
104	VictorAudio24-6.wav	15	4	15	15	15
105	VictorAudio24-9.wav	9 10	10	10	10	10
106	VictorAudio24-10.wav	17	3	1	17	1
107	VictorAudio24-11.wav	18	3	1	18	1
108	VictorAudio24-12.wav	6	6	6	6	6
109	VictorAudio24-13.wav	None	10	1	1	1
110	VictorAudio25-0.wav	0 1	1	1	1	1
111	VictorAudio25-1.wav	25	0	25	25	25
112	VictorAudio25-2.wav	18	0	8	8	8
113	VictorAudio25-3.wav	None	0	16	25	16
114	VictorAudio25-4.wav	17	0	17	17	17
115	VictorAudio25-5.wav	6	1	6	6	6
116	VictorAudio25-6.wav	22	22	22	22	22
117	VictorAudio25-7.wav	24	0	24	24	24
118	VictorAudio25-8.wav	19	19	19	19	19
119	VictorAudio25-9.wav	None	0	25	25	25
120	VictorAudio26-1.wav	0 1	1	1	1	1
121	VictorAudio26-2.wav	19	1	19	19	19
122	VictorAudio26-3.wav	22 23	22	23	23	23
123	VictorAudio26-4.wav	14	14	14	14	14
124	VictorAudio26-5.wav	4 5	4	4	4	4
125	VictorAudio26-6.wav	17	17	25	25	25
126	VictorAudio26-7.wav	6	0	6	6	6
127	VictorAudio27-0.wav	0	0	0	0	0
128	VictorAudio27-1.wav	3	3	1	1	1
129	VictorAudio27-2.wav	19	19	19	19	19
130	VictorAudio27-3.wav	25	0	25	25	25
131	VictorAudio27-4.wav	None	0	25	25	25
132	VictorAudio27-5.wav	None	4	25	25	25
133	VictorAudio27-6.wav	15	8	15	15	15
134	VictorAudio27-7.wav	18	1	18	18	18
135	VictorAudio28-0.wav	0 1	1	1	1	1
136	VictorAudio28-1.wav	None	0	6	6	6

#	Wave File	Template	Full	Linear	SIDF	IDF
137	VictorAudio28-2.wav	8	8	8	8	8
138	VictorAudio28-3.wav	25	1	1	15	1
139	VictorAudio29-0.wav	9 10	10	1	1	1
140	VictorAudio29-1.wav	None	0	1	1	1
141	VictorAudio29-2.wav	None	0	1	1	1
142	VictorAudio29-3.wav	None	0	1	1	1
143	VictorAudio29-4.wav	None	0	10	25	10
144	VictorAudio29-5.wav	6	6	1	1	1
145	VictorAudio29-7.wav	None	0	1	1	1
146	VictorAudio29-8.wav	None	4	15	15	15
147	VictorAudio29-9.wav	None	0	17	1	17
148	VictorAudio29-10.wav	None	0	1	6	1
149	VictorAudio29-11.wav	None	8	1	18	1
150	VictorAudio29-12.wav	None	22	21	21	21
151	VictorAudio29-13.wav	None	0	19	19	19
152	VictorAudio29-14.wav	None	0	8	8	8
153	VictorAudio29-15.wav	11 12	11	11	11	11
154	VictorAudio29-16.wav	4	4	4	4	4
155	VictorAudio29-17.wav	15	15	15	15	15
156	VictorAudio29-18.wav	16	24	24	16	24
157	VictorAudio29-19.wav	18	3	18	18	18
158	VictorAudio29-20.wav	16	0	16	16	16
159	VictorAudio29-21.wav	3	0	3	3	3
160	VictorAudio29-22.wav	4	0	4	4	4
161	VictorAudio29-23.wav	None	0	10	10	10
162	VictorAudio29-24.wav	25	0	25	25	25
163	VictorAudio29-25.wav	None	0	7	7	7
164	VictorAudio29-26.wav	None	0	15	15	15
165	VictorAudio29-27.wav	None	0	14	14	14
166	VictorAudio29-28.wav	None	1	1	6	1
167	VictorAudio29-29.wav	3	0	3	3	3
168	VictorAudio29-30.wav	None	0	8	8	8

Final Summary of Test Data Analysis

For each of the reasonable questions, an indication of success for each weighting method is shown. A "1" indicates that the weighting method responded properly (as a first choice). A "0" indicates that the method did not choose appropriately.

Final Counts

Full Question Total = 124
 Component Linear Total = 204
 Component SIDF Total = 209
 Component IDF Total = 205
 Component Binary Total = 182

Table 9. Data Counts

Wave File	Full	Linear	SIDF	IDF	Binary
FredAudio6-0.wav	1	0	0	0	1
FredAudio6-1.wav	0	1	1	1	0
FredAudio6-2.wav	1	1	1	1	1
FredAudio6-3.wav	1	1	1	1	1
FredAudio6-4.wav	1	1	1	1	1
FredAudio6-5.wav	1	1	1	1	1
FredAudio6-6.wav	1	1	1	1	1
FredAudio6-7.wav	1	1	0	1	1
FredAudio6-8.wav	1	1	1	1	1
FredAudio6-9.wav	1	1	1	1	1
FredAudio6-10.wav	1	1	1	1	1
FredAudio6-11.wav	0	0	1	0	0
FredAudio6-12.wav	0	0	1	0	0
FredAudio6-15.wav	0	1	1	1	1
FredAudio6-22.wav	1	1	1	1	1
FredAudio6-24.wav	1	1	1	1	1
FredAudio6-28.wav	0	0	1	0	0
FredAudio7-0.wav	1	1	1	1	1
FredAudio7-1.wav	1	1	1	1	1
FredAudio7-3.wav	1	1	1	1	1
FredAudio7-8.wav	0	1	1	1	1
FredAudio7-15.wav	1	0	1	0	0
FredAudio8-0.wav	1	1	1	1	1
FredAudio8-1.wav	0	1	1	1	1
FredAudio8-2.wav	0	1	1	1	1
FredAudio8-5.wav	1	1	0	1	1
FredAudio8-7.wav	0	1	1	1	0
FredAudio8-8.wav	0	1	1	1	1

Wave File	Full	Linear	SIDF	IDF	Binary
FredAudio8-10.wav	0	0	0	0	0
FredAudio8-11.wav	1	1	0	1	0
FredAudio8-12.wav	0	1	1	1	0
FredAudio8-13.wav	0	0	1	0	0
FredAudio8-14.wav	1	1	1	1	1
FredAudio8-18.wav	0	1	1	1	1
FredAudio8-21.wav	0	0	0	0	0
FredAudio8-22.wav	1	1	1	1	1
FredAudio8-23.wav	0	1	1	1	1
FredAudio9-0.wav	1	1	0	1	1
FredAudio9-1.wav	0	0	0	0	0
FredAudio9-3.wav	1	1	1	1	1
FredAudio9-4.wav	0	1	1	1	1
FredAudio9-7.wav	0	1	1	1	1
FredAudio9-9.wav	1	1	0	1	1
FredAudio9-10.wav	1	1	1	1	1
FredAudio9-11.wav	0	1	1	1	1
FredAudio9-12.wav	0	1	1	1	1
FredAudio9-13.wav	1	1	1	1	1
FredAudio9-15.wav	0	1	1	1	1
FredAudio9-16.wav	1	1	1	1	1
FredAudio9-17.wav	0	1	1	1	0
FredAudio9-18.wav	0	0	0	0	0
FredAudio9-19.wav	1	1	1	1	1
FredAudio9-20.wav	1	1	1	1	1
FredAudio9-21.wav	1	1	0	1	1
FredAudio9-22.wav	0	1	1	1	0
FredAudio10-0.wav	1	1	1	1	0
FredAudio10-1.wav	0	1	1	1	1
FredAudio10-2.wav	1	1	1	1	1
FredAudio10-3.wav	1	0	0	0	0
FredAudio10-4.wav	0	0	0	0	0
FredAudio10-5.wav	0	1	1	1	0
FredAudio10-6.wav	0	0	1	0	0
FredAudio10-7.wav	0	1	1	1	0
FredAudio10-8.wav	1	0	1	0	0
FredAudio10-9.wav	1	1	1	1	1
FredAudio10-10.wav	1	0	0	0	0
FredAudio10-11.wav	1	1	1	1	1
FredAudio10-12.wav	1	1	1	1	1
FredAudio10-13.wav	1	1	1	1	1
FredAudio10-14.wav	1	1	1	1	1
FredAudio10-15.wav	1	1	1	1	1
FredAudio10-16.wav	1	1	1	1	1
FredAudio10-17.wav	1	1	1	1	0

Wave File	Full	Linear	SIDF	IDF	Binary
FredAudio10-19.wav	1	1	1	1	1
FredAudio10-20.wav	1	1	1	1	0
VictorAudio14-0.wav	0	0	0	0	0
VictorAudio14-1.wav	1	1	1	1	1
VictorAudio14-2.wav	0	0	0	0	0
VictorAudio14-7.wav	1	1	1	1	1
VictorAudio14-9.wav	0	1	1	1	0
VictorAudio14-10.wav	1	1	1	1	1
VictorAudio14-12.wav	0	1	1	1	1
VictorAudio14-13.wav	0	1	1	1	1
VictorAudio15-0.wav	1	1	1	1	1
VictorAudio15-1.wav	1	1	1	1	1
VictorAudio15-6.wav	0	0	1	0	0
VictorAudio15-8.wav	1	1	1	1	1
VictorAudio15-9.wav	1	1	1	1	1
VictorAudio15-10.wav	0	1	1	1	1
VictorAudio15-12.wav	0	1	1	1	1
VictorAudio15-13.wav	0	0	0	0	0
VictorAudio15-14.wav	0	1	1	1	1
VictorAudio15-15.wav	0	1	1	1	1
VictorAudio15-16.wav	0	1	1	1	1
VictorAudio15-17.wav	0	1	1	1	1
VictorAudio15-18.wav	0	1	1	1	1
VictorAudio15-19.wav	0	1	1	1	1
VictorAudio15-20.wav	1	1	1	1	1
VictorAudio15-21.wav	1	1	1	1	1
VictorAudio15-22.wav	0	1	1	1	1
VictorAudio15-25.wav	0	0	0	0	0
VictorAudio15-26.wav	0	1	1	1	1
VictorAudio15-27.wav	0	1	1	1	1
VictorAudio15-28.wav	0	1	1	1	1
VictorAudio16-0.wav	1	1	0	1	1
VictorAudio16-1.wav	0	0	0	0	0
VictorAudio16-2.wav	0	1	1	1	1
VictorAudio16-3.wav	1	1	1	1	1
VictorAudio16-4.wav	1	1	1	1	1
VictorAudio16-5.wav	1	1	1	1	1
VictorAudio16-6.wav	1	1	1	1	0
VictorAudio16-7.wav	1	1	1	1	1
VictorAudio16-8.wav	1	1	0	1	1
VictorAudio16-9.wav	0	1	1	1	1
VictorAudio16-10.wav	0	1	1	1	1
VictorAudio16-11.wav	0	1	1	1	1
VictorAudio16-12.wav	0	1	1	1	1
VictorAudio16-13.wav	0	1	1	1	1

Wave File	Full	Linear	SIDF	IDF	Binary
VictorAudio16-14.wav	1	1	1	1	0
VictorAudio16-16.wav	0	1	1	1	1
VictorAudio17-0.wav	1	1	0	1	1
VictorAudio17-2.wav	0	1	1	1	1
VictorAudio17-3.wav	0	0	0	0	0
VictorAudio17-4.wav	0	1	1	1	1
VictorAudio17-5.wav	1	1	1	1	1
VictorAudio17-10.wav	1	1	1	1	1
VictorAudio18-0.wav	0	0	0	0	0
VictorAudio18-1.wav	0	1	1	1	1
VictorAudio18-7.wav	0	0	0	0	0
VictorAudio18-8.wav	0	1	1	1	1
VictorAudio18-9.wav	0	1	1	1	1
VictorAudio18-10.wav	1	0	0	0	1
VictorAudio18-11.wav	1	1	0	1	1
VictorAudio18-12.wav	0	0	0	0	0
VictorAudio18-13.wav	1	1	1	1	1
VictorAudio18-14.wav	1	1	1	1	1
VictorAudio18-17.wav	1	1	1	1	1
VictorAudio18-21.wav	0	0	0	0	0
VictorAudio18-22.wav	0	0	0	0	0
VictorAudio18-23.wav	1	1	1	1	1
VictorAudio19-0.wav	1	1	1	1	1
VictorAudio19-1.wav	0	1	1	1	1
VictorAudio19-2.wav	0	1	1	1	1
VictorAudio19-3.wav	0	1	1	1	1
VictorAudio19-4.wav	0	1	1	1	1
VictorAudio19-5.wav	0	0	0	0	0
VictorAudio19-6.wav	0	0	0	0	0
VictorAudio19-7.wav	0	0	0	0	0
VictorAudio19-8.wav	0	1	1	1	1
VictorAudio19-9.wav	0	1	1	1	1
VictorAudio19-10.wav	0	0	0	0	0
VictorAudio19-11.wav	0	1	1	1	1
VictorAudio19-12.wav	0	1	1	1	1
VictorAudio19-15.wav	0	1	1	1	1
VictorAudio19-16.wav	0	0	0	0	0
VictorAudio19-17.wav	0	1	1	1	1
VictorAudio19-19.wav	0	0	0	0	0
VictorAudio19-20.wav	0	1	1	1	1
VictorAudio19-24.wav	0	1	1	1	1
VictorAudio19-27.wav	0	1	1	1	1
VictorAudio19-33.wav	0	1	1	1	1
VictorAudio19-35.wav	0	1	1	1	1
VictorAudio19-38.wav	0	0	0	0	0

Wave File	Full	Linear	SIDF	IDF	Binary
VictorAudio19-39.wav	0	0	0	0	0
VictorAudio19-50.wav	0	1	1	1	1
VictorAudio19-55.wav	0	1	1	1	1
VictorAudio19-56.wav	0	1	1	1	1
VictorAudio19-65.wav	0	1	1	1	1
VictorAudio20-0.wav	1	1	1	1	1
VictorAudio20-1.wav	0	1	1	1	0
VictorAudio20-3.wav	0	1	1	1	1
VictorAudio20-4.wav	1	1	1	1	1
VictorAudio20-6.wav	0	0	0	0	0
VictorAudio20-7.wav	0	1	1	1	1
VictorAudio20-8.wav	0	0	0	0	0
VictorAudio20-9.wav	1	0	0	0	0
VictorAudio20-10.wav	1	1	0	1	1
VictorAudio20-11.wav	1	1	1	1	1
VictorAudio20-12.wav	0	1	1	1	1
VictorAudio20-13.wav	0	0	0	0	0
VictorAudio20-14.wav	0	0	0	0	0
VictorAudio20-15.wav	0	1	1	1	1
VictorAudio20-16.wav	1	0	0	0	0
VictorAudio20-17.wav	0	0	0	0	0
VictorAudio20-19.wav	0	1	1	1	1
VictorAudio21-0.wav	1	0	0	0	1
VictorAudio21-1.wav	1	1	1	1	1
VictorAudio21-2.wav	1	1	1	1	1
VictorAudio21-3.wav	1	1	1	1	1
VictorAudio21-4.wav	1	1	1	1	1
VictorAudio21-5.wav	1	1	1	1	1
VictorAudio21-6.wav	1	1	1	1	1
VictorAudio21-7.wav	1	1	1	1	0
VictorAudio21-8.wav	1	1	1	1	1
VictorAudio21-9.wav	1	1	1	1	1
VictorAudio21-10.wav	0	0	1	0	0
VictorAudio21-11.wav	1	0	1	0	0
VictorAudio22-2.wav	0	1	1	1	1
VictorAudio22-4.wav	1	0	0	0	0
VictorAudio22-6.wav	1	1	1	1	0
VictorAudio22-9.wav	1	1	1	1	1
VictorAudio22-14.wav	1	1	1	1	1
VictorAudio22-17.wav	1	0	1	1	0
VictorAudio22-25.wav	0	0	0	0	0
VictorAudio22-32.wav	0	1	1	1	0
VictorAudio22-34.wav	0	1	1	1	0
VictorAudio22-35.wav	0	1	1	1	1
VictorAudio22-36.wav	0	0	1	0	0

Wave File	Full	Linear	SIDF	IDF	Binary
VictorAudio22-37.wav	0	1	1	1	0
VictorAudio22-49.wav	0	1	1	1	1
VictorAudio22-52.wav	0	1	1	1	1
VictorAudio22-55.wav	1	0	0	0	0
VictorAudio23-0.wav	1	1	1	1	1
VictorAudio23-1.wav	0	1	1	1	1
VictorAudio23-2.wav	1	1	1	1	1
VictorAudio23-5.wav	1	1	1	1	1
VictorAudio23-6.wav	0	0	0	0	0
VictorAudio23-7.wav	0	1	1	1	1
VictorAudio23-8.wav	1	1	1	1	1
VictorAudio23-9.wav	0	1	1	1	0
VictorAudio23-10.wav	0	0	0	0	0
VictorAudio23-11.wav	0	0	1	0	0
VictorAudio24-0.wav	1	0	0	0	1
VictorAudio24-1.wav	1	1	1	1	1
VictorAudio24-2.wav	1	1	1	1	1
VictorAudio24-3.wav	1	1	1	1	1
VictorAudio24-4.wav	1	1	1	1	1
VictorAudio24-6.wav	0	1	1	1	1
VictorAudio24-9.wav	1	1	1	1	0
VictorAudio24-10.wav	0	0	1	0	0
VictorAudio24-11.wav	0	0	1	0	0
VictorAudio24-12.wav	1	1	1	1	1
VictorAudio25-0.wav	1	1	1	1	1
VictorAudio25-1.wav	0	1	1	1	0
VictorAudio25-2.wav	0	0	0	0	0
VictorAudio25-4.wav	0	1	1	1	0
VictorAudio25-5.wav	0	1	1	1	1
VictorAudio25-6.wav	1	1	1	1	1
VictorAudio25-7.wav	0	1	1	1	1
VictorAudio25-8.wav	1	1	1	1	0
VictorAudio26-1.wav	1	1	1	1	1
VictorAudio26-2.wav	0	1	1	1	1
VictorAudio26-3.wav	1	1	1	1	1
VictorAudio26-4.wav	1	1	1	1	1
VictorAudio26-5.wav	1	1	1	1	1
VictorAudio26-6.wav	1	0	0	0	0
VictorAudio26-7.wav	0	1	1	1	1
VictorAudio27-0.wav	1	1	1	1	1
VictorAudio27-1.wav	1	0	0	0	0
VictorAudio27-2.wav	1	1	1	1	0
VictorAudio27-3.wav	0	1	1	1	1
VictorAudio27-6.wav	0	1	1	1	1
VictorAudio27-7.wav	0	1	1	1	1

Wave File	Full	Linear	SIDF	IDF	Binary
VictorAudio28-0.wav	1	1	1	1	1
VictorAudio28-2.wav	1	1	1	1	1
VictorAudio28-3.wav	0	0	0	0	0
VictorAudio29-0.wav	1	0	0	0	0
VictorAudio29-5.wav	1	0	0	0	0
VictorAudio29-15.wav	1	1	1	1	1
VictorAudio29-16.wav	1	1	1	1	1
VictorAudio29-17.wav	1	1	1	1	1
VictorAudio29-18.wav	0	0	1	0	0
VictorAudio29-19.wav	0	1	1	1	1
VictorAudio29-20.wav	0	1	1	1	1
VictorAudio29-21.wav	0	1	1	1	1
VictorAudio29-22.wav	0	1	1	1	1
VictorAudio29-24.wav	0	1	1	1	1
VictorAudio29-29.wav	0	1	1	1	1

APPENDIX B

CD CONTENTS

The CD (Compact Disc) that accompanies this document contains seven folders.

- AudioFiles
- DataFiles
- ExecutableCode
- SAMSetup
- SourceCode
- TestMaterials
- Thesis

AudioFiles

This folder contains all of the audio files that were captured from test subjects. It has four subdirectories: Test1Audio, Test2Audio, Test3Audio, and Test4Audio. The files in Test1Audio were used in the optimization of the system. The other three folders contain files used in the final data collection. The files were split into 3 groups for processing.

DataFiles

This folder contains compiled test data used in the system optimization and final test. The documents Test1.txt, and TestData.txt contain a block of text for each test question as described in Chapter 8, Data Collection. The folder also contains a file of summary information for each data file as explained in Chapter 9, Analysis.

ExecutableCode

This folder contains projects that run. It has five subdirectories.

- FredComponent
- FredFullQuestion
- FredLogger
- Student
- Ted

The first four are all Fred type systems. They each have a Fred.exe file. Double clicking this file will launch the application. They each have a grammar file. They each have a record file.

FredComponent contains the driver record file that was used in the testing of the system. It has a grammar file that instructs Fred to use Component recognition.

FredFullQuestion contains the same record file, but uses a full question grammar file.

FredLogger is another version of Fred identical to the normal version, except it writes data into a log file as test questions are asked. This is the version that was used in data collection and processing. It contains a driver record, and both types of grammar files.

Student contains the standard version of Fred. The record file is a student record file. The grammar file is a Student system with full question recognition. The folder also contains a transcript corresponding to the record file, and the Ted project file.

Ted contains the source and executable for the Ted application. Start the program by double clicking on the Ted shortcut.

SAMSetup

This folder contains the files required to install Fred on a computer system. This includes the Microsoft SAPI 5 speech SDK, and files to install the UNH SAPI interface files. The folder contains a document called Setup.txt that outlines the setup process.

SourceCode

This folder contains all of the source code used in the research. It contains five subdirectories.

- AnalysisTools
- DriverRecordParser
- Fred
- FredLogger
- Ted

The AnalysisTools folder contains several Java programs written to analyze the test data. The Merger program merges together the component and full question portions of the log files. The Analyzer program reads in a merged edited data file, as explained in chapter 9, Analysis, and writes a report summarizing the test data. Both programs are written in standard Java, and can be edited or launched using any Java IDE.

The DriverRecordParser folder contains a program that parses New Hampshire diver records. The parser writes a record file in the proper format for Fred, as described in chapter 4, Using Fred. The program is written in C, and is part of a Microsoft Visual C++ project. The project can be opened by double clicking on the DRParser.dsw file. A built executable is stored in the Debug folder. The application can be launched by double clicking the DRParser.exe file.

The Fred folder contains all of the source code for Fred as described in chapter 5, How Fred Works. The program is written in C, and is part of a

Microsoft Visual C++ project. The project can be opened by double clicking on the Fred.dsw file.

The FredLogger folder contains the source code for a version of Fred that writes data into a log file as test questions are asked. The program is written in C, and is part of a Microsoft Visual C++ project. The project can be opened by double clicking on the Victor.dsw file.

Ted contains the source and executable for the Ted application. The program is written in Java, and is part of a Microsoft Visual C++ project. The project can be opened by double clicking on the Ted.sln file.

Test Materials

This folder contains the documents that test subjects were allowed to see. Ask Fred.doc is the instruction sheet described in chapter 8, Data Collection. Driver Record Tree.doc is the tree diagram, also described in chapter 8, Data Collection.

Thesis

This folder contains all of the chapters and appendices of the thesis.

APPENDIX C

RELEASE FORM

Each subject signed a copy of the form shown on the following pages.

The form was supplied by the UNH Institutional Review Board. The signed forms were faxed to the IRB.

UNIVERSITY OF NEW HAMPSHIRE
INSTITUTIONAL REVIEW BOARD FOR THE PROTECTION OF HUMAN SUBJECTS IN RESEARCH

Purpose: The purpose of this research is to assist in the development of speech user interfaces as well as other user interfaces for mobile environments such as vehicles and handheld computers. Another goal is to develop specific applications for mobile environments, specifically for vehicles and for places where people use handheld computers.

Procedure: ■ computer. The Project54 system will record your speech, and/or your interactions with the GUI and/or your interactions with original hardware interfaces, and/or data generated by electronic devices that you interact with and/or data generated by electronic devices that the Project54 system interacts with. The recording will require no special steps on your part. You will be asked to interact with the Project54 system running on a PC and/or on a handheld devices. We will create audio and/or video recordings of your interactions. We will also record your interactions with the computer's GUI and/or your interactions with other hardware interfaces, and/or data generated by the computer and/or by the electronic devices. You will be asked to interact with a PC and/or on a handheld computer and/or other electronic

Data generated in this research will be saved for use in future research. A unique ID will be assigned to you. The unique ID will be of the form "User #xx", where xx is the number assigned to you. It will be used to label your data, along with your age, gender, characteristics of your speech, your experience in working with computers or the Project54 system and any questionnaires you fill out. The data will be stored for future use in our research. Your identity will not be tied to the data in any way (other than to the video data, if such data is created, since video data may visually identify you). In this document we are asking for your consent to participate in our study and to share the non-video data with researchers from other institutions. Separately we also ask for your consent to share video data with researchers from other institutions as well as to show video data at conferences and similar meetings.

This research should present no risk to you. There should be no aftereffects of this research upon you. There will be no monetary compensation for your work.

1. You understand that the use of human subjects in this project has been approved by the UNH Institutional Review Board for the Protection of Human Subjects in Research.
2. You understand the scope, aims, and purposes of this research project and the procedures to be followed and the expected duration of your participation.
3. You have received a description of any reasonable foreseeable risks or discomforts associated with being a subject in this research, have had them explained to you, and understand them.
4. You have received a description of any potential benefits that may be accrued from this research and understand how they may affect you or others.
5. The investigator seeks to maintain the confidentiality of all data and records associated with your participation in this research. You should understand, however, there are rare instances when the investigator is required to share personally-identifiable information (e.g., according to policy, contract, regulation). For example, in response to a complaint about the research, officials at the University of New Hampshire, designees of the sponsor(s), and/or regulatory and oversight government agencies may access research data.

6. You understand that your consent to participate in this research is entirely voluntary, and that your refusal to participate will involve no prejudice, penalty or loss of benefits to which you would otherwise be entitled.

7. You further understand that if you consent to participate, you may discontinue your participation at any time without prejudice, penalty, or loss of benefits to which you would otherwise be entitled.

Office of Sponsored Research - Regulatory Compliance/Phone: 862-2003 Rev. 8/01

8. You confirm that no coercion of any kind was used in seeking your participation in this research project.

9. You understand that if you have any questions pertaining to the research you can call Dr. Andrew Kun at 603-862-4175 and be given the opportunity to discuss them. If you have questions pertaining to your rights as a research subject you can call Julie Simpson in the UNH Office of Sponsored Research, 603-862-2003, to discuss them.

10. You understand that you will not be provided financial incentive for your participation by the University of New Hampshire.

11. You understand that your age, gender, the characteristics of your speech, and your experience in working with computers or the Project54 system will be recorded, and may be shared with other researchers, along with the data collected about your interactions.

12. You certify that you have read and fully understand the purpose of this research project and the risks and benefits it presents to you as stated above.

I, _____

CONSENT/AGREE to participate in this research project.

I, _____

REFUSE/DO NOT AGREE to participate in this research project.

Signature of Subject

Date

I, _____

CONSENT/AGREE to allow sharing video data with other researchers and showing it at conferences and similar meetings.

I, _____

REFUSE/DO NOT AGREE to allow sharing video data with other researchers or showing it at conferences and similar meetings.

Signature of Subject

Date

APPENDIX D

IRB APPROVAL

This research was done in conjunction with another research project, and was given approval under that project. The letter below demonstrates compliance to the requirements as outlined in the Graduate School's Thesis and Dissertation Manual.

University of New Hampshire

Research Conduct and Compliance Services, Office of Sponsored Research
Service Building, 51 College Road, Durham, NH 03824-3585
Fax: 603-862-3564

01-Nov-2001

Kun, Andrew
Electrical & Computer Eng Dept
Kingsbury Hall
Durham, NH 03824

IRB #: 2980

Study: Speech Sample Collection for Speech Recognition Engine Comparison and Development

Approval Expiration Date: 24-Jun-2008

Modification Approval Date: 31-Oct-2001

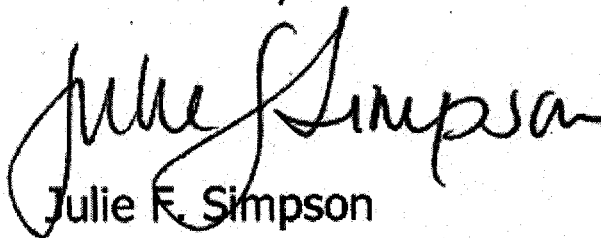
Modification: Collection of additional data (e.g. physiological measures) per 10/22/2001 email. The Institutional Review Board for the Protection of Human Subjects in Research (IRB) has reviewed and approved your modification to this study, as indicated above. Further changes in your study must be submitted to the IRB for review and approval prior to implementation.

Approval for this protocol expires on the date indicated above. At the -end of the approval period you will be asked to submit a report with regard to the involvement of human subjects in this study. If your study is still active, you may request an extension of IRB approval.

Researchers who conduct studies involving human subjects have responsibilities as outlined in the document, *Responsibilities of Directors of Research Studies Involving Human Subjects*. This document is available at <http://www.unh.edu/osr/compliance/irb.html> or from me.

If you have questions or concerns about your study or this approval, please feel free to contact me at 603-862-2003 or Julie.simpson@unh.edu. Please refer to the IRB # above in all correspondence related to this study. The IRB wishes you success with your research.

For the IRB,



Julie F. Simpson
Manager

cc: File

APPENDIX E

ACKNOWLEDGEMENT OF FUNDING

This work was supported in part by the U.S. Department of Justice under grants 1999-DD-BX-0082 and 2001-LT-BX-K010.

APPENDIX F

SOFTWARE TOOLS

The software tools used in the research described in this dissertation were originally given "internal" names. As such, this documentation refers to each of the tools by these names. SAM refers to the entire system developed to create, edit, and run spoken question answering systems. The SAM system has two components. The editing component is referred to as Ted. The runtime component is referred to as Fred.

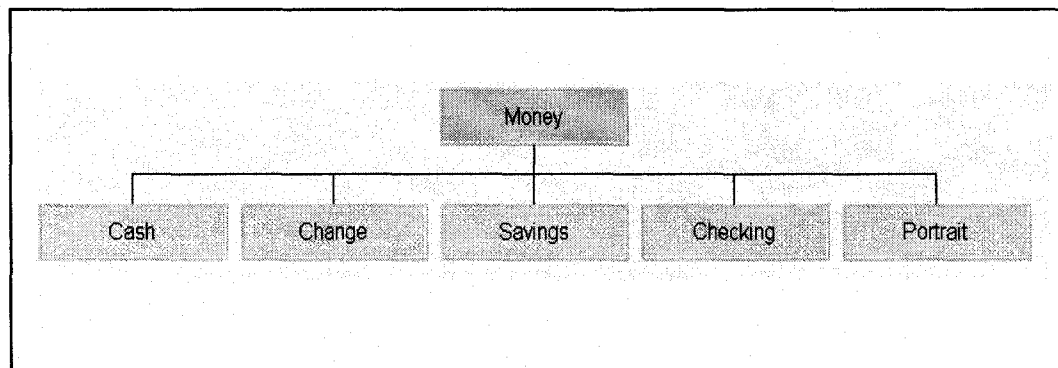
Using Ted

Ted is an editing application intended to be used as part of the SAM Q/A system. It generates files to be used with its runtime counterpart Fred to create a spoken question answering system. Ted creates and edits Ted (*.ted) files, which represent information trees. An information tree has a leaf node for each piece of information. In general, there may be branch nodes as well. Every tree has at least one branch node (commonly referred to as the root node), which is the top node of the tree. The root node can create (or be the parent node of), any number of branch or leaf nodes (child nodes). However, a leaf node can not create nodes (a leaf node can not have children). A leaf node represents the end of a branch. Leaf nodes contain information. Branch nodes are used only for organization.

So, a simple tree for your money might include leaf nodes for: the cash in your pocket, the change on your dresser, the money in your savings account, the money in your checking account, and the money you keep hidden behind the second portrait in the hallway.

Figure 9 below depicts a Ted type tree for a money system as described above.

Figure 9. Money Tree Diagram



Here, the node "Money" is a branch node, and it has five child nodes. The child nodes are all leaf nodes. The branch node has no information associated with it. However, each leaf node is associated with a number, the amount of money in that place.

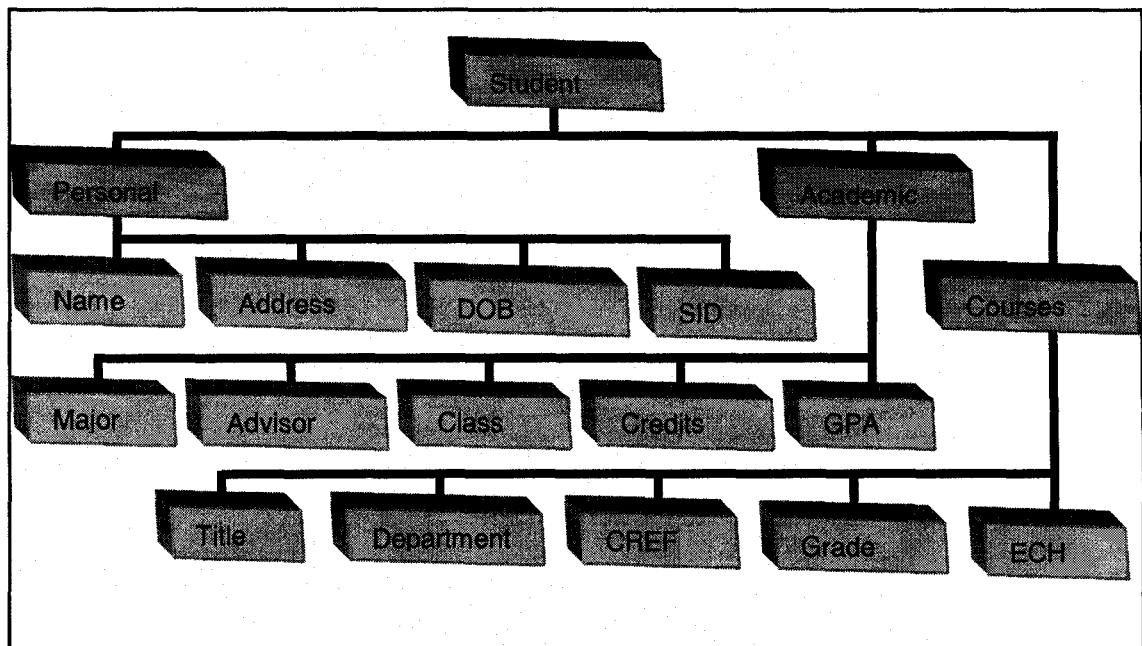
It is important to point out here, that Ted has nothing to do with this information or the storage of it. Ted allows the developer to define a structure that has leaf nodes that correspond to information in some external record. It is assumed that at least one such record exists. There may be more than one record. For example, Money records might exist for multiple people.

What Ted Does

A data record can have a complex structure with many pieces of data. Some data fields may have multiple values. Strictly speaking, Ted is designed to work with relational database queries that have been stored as text files. For the purposes of this research, this file is considered, "the record". Ted allows you to create a structure that is compatible with a type of record, and enter sample questions about the record information. Ted then generates a file that is used with its counterpart Fred. Together, Ted and Fred form a system that reads records, and responds to spoken questions. Which questions the system will respond to, and how it responds to them are defined within Ted.

Take for example a student record such as one would find on a college banner system. The record structure could be depicted in Ted as shown in Figure 10 below.

Figure 10. Student Tree Diagram



Note that while some fields have only one value associated with them (Address, or Credits), others may have many values (Grade). Ted offers a simple scripting language that searches the record, and returns a natural response to a question. For the student record, the Ted file might include questions such as these.

Does the student have a major?

Who is the student's advisor?

What year is the student in?

What class is the student in?

How many credits does the student have?

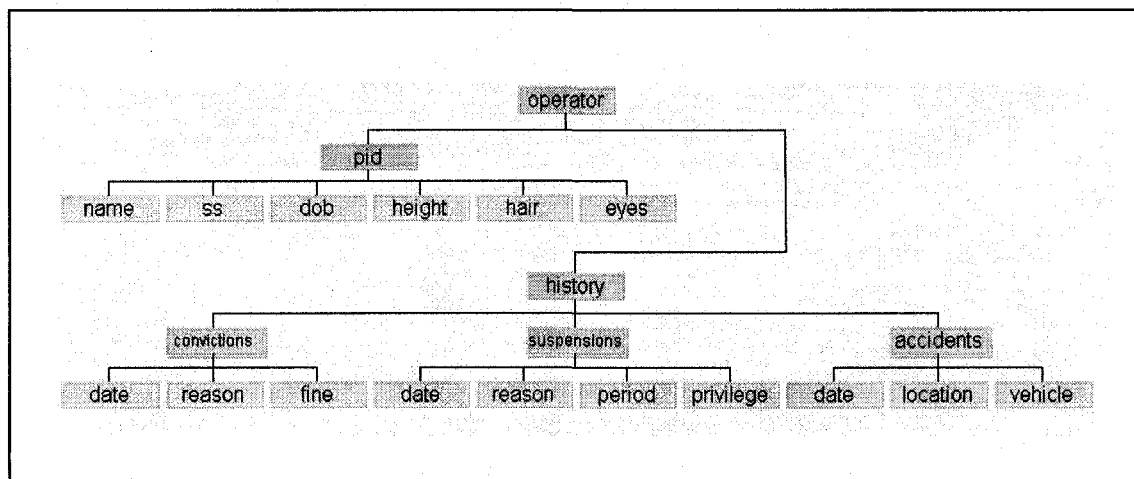
Does the student have enough credits to graduate?

How many courses has the student passed?

Another Example

As a second example, consider the driver record tree diagram shown in figure 11.

Figure 11. Driver Record Tree Diagram



Ted is used to organize questions and associated answer scripts. Questions about the operator's social security number might go in the "ss" node. A more general question that uses more information than that from a single leaf, "What does the operator look like?" might go in the "pid" node. Note that there may be multiple values for items under the convictions, suspensions, and accidents nodes. Each conviction entry has a date, reason, and fine associated with it. Ted allows you to ask questions about the entire collection, or a subset like "What was the operator convicted of in 1998?" To use the more advanced features of Ted, you will need to finish reading this chapter, but you should now have an understanding of what Ted does.

Opening and Closing Ted

To open Ted, use a Java virtual machine like jview. To open Ted on a PC, open a command prompt, navigate to the Ted directory, and type:

```
jview /a TED.htm
```

If the command prompt is closed, Ted will exit immediately. You can also launch Ted by double clicking the Ted shortcut. To close Ted, simply close the window that Ted is in (click the x in the upper right corner). **Important:** When Ted closes, you will **not** be asked if you want to save your current file. If your file closes because you close Ted, open or create a new Ted file, or close the jview window, any unsaved information will be lost.

The Node Properties Dialog Boxes

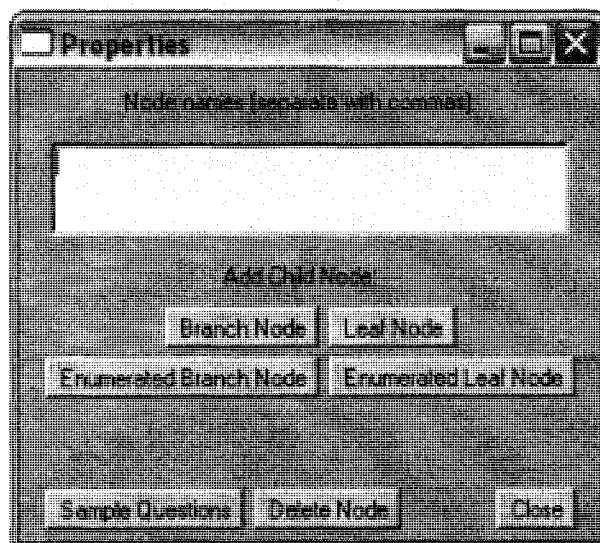
Each node has an associated Properties dialog box. To view this dialog box, simply click on the node. To avoid confusion, Ted will not allow more than one node Properties dialog box to be open at any time. If at any time you are unable to open a node Properties dialog box, check to see if one is already opened in the background. To close a node Properties dialog box, click the "Close" button, or close the window.

There are two types of node Properties dialog boxes: the branch node Properties dialog box, and the leaf node Properties dialog box.

The Branch Node Properties Dialog Box

If you click on a branch node, a window like that shown in figure 12 will open.

Figure 12. Branch Node Properties Dialog Box



The text box at the top holds the name(s) of the node. These names should be separated by commas. Spaces are not required. If a node has more than one name, the other names will be treated as synonyms for the first name if it is used in a question. This is one way Ted allows you to make your system relate more specifically to your environment.

The names may consist of a sequence of any characters or spaces with the following reservations. The first name may contain no spaces, and all first node names must be unique. The first node name is the one that will appear in the tree diagram.

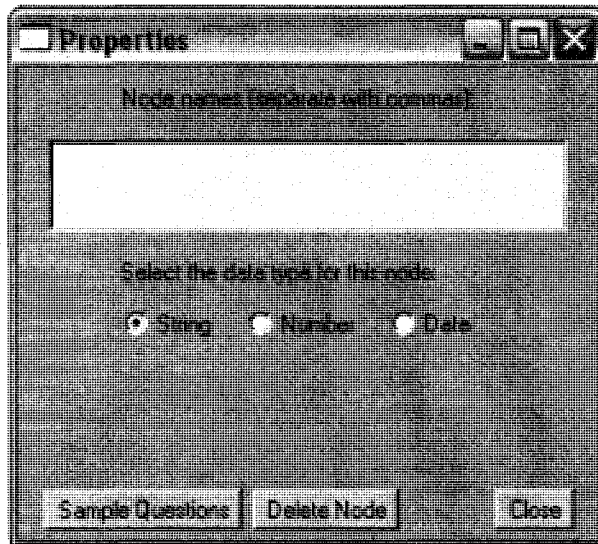
Beneath the Node Names textbox are the Add Child Node buttons. You may click any of these to create a new node that will be connected as a child to the current node. Enumerated nodes are meant to hold multiple values (or sets of values). However, Ted does not distinguish between normal and enumerated nodes. The enumerated node buttons have been retained only so the application will work with earlier Ted files.

The buttons on the bottom of the dialog box are used to view and edit the sample questions (discussed later in this chapter), delete the current node and all child nodes, and close the dialog box.

The Leaf Node Properties Dialog Box

If you click on a leaf node, a window like that shown in figure 13 will open.

Figure 13. Leaf Node Properties Dialog Box



This dialog box operates the same as the branch node Properties dialog box. The only difference is that a leaf node can not create child nodes. Instead, each leaf node has an associated data type. Select the data type by clicking one of the three radio buttons.

The Menu

Ted has a number of useful features that are accessible through the menus.

These menus are discussed, in order, below.

File

New File. Choosing this option clears Ted's memory and starts a new tree. All new trees begin with one empty branch node (the root node). Be sure to save your current project first. If you select New File, any unsaved information will be lost.

Open File. Choosing this option opens the "Select File to Open" dialog box. Here, you can browse for an existing Ted file. Be sure to save your current project first. If you select Open File, any unsaved information will be lost.

Save As. Choosing this option opens the "Save As" dialog box. Here, you can browse for a location in which to save the current file, and choose a file name. By default, the extension .ted is suggested. This extension is not required, but it makes your Ted files easier to find.

Zoom

The Zoom menu allows you to look at a small group of nodes, or back up and view larger sections of the tree. As the nodes get smaller, their text will also get smaller. When the text becomes too small to be reasonably legible, it is omitted. See the next section on tags. The Zoom menu is always set to one of the following four options.

1:1. This is the normal viewing ratio.

1:2. Choosing this option makes everything half sized.

1:4. Choosing this option makes everything quarter sized.

1:8. Choosing this option makes everything eighth sized.

View

The View menu offers two options, and must be set to one or the other.

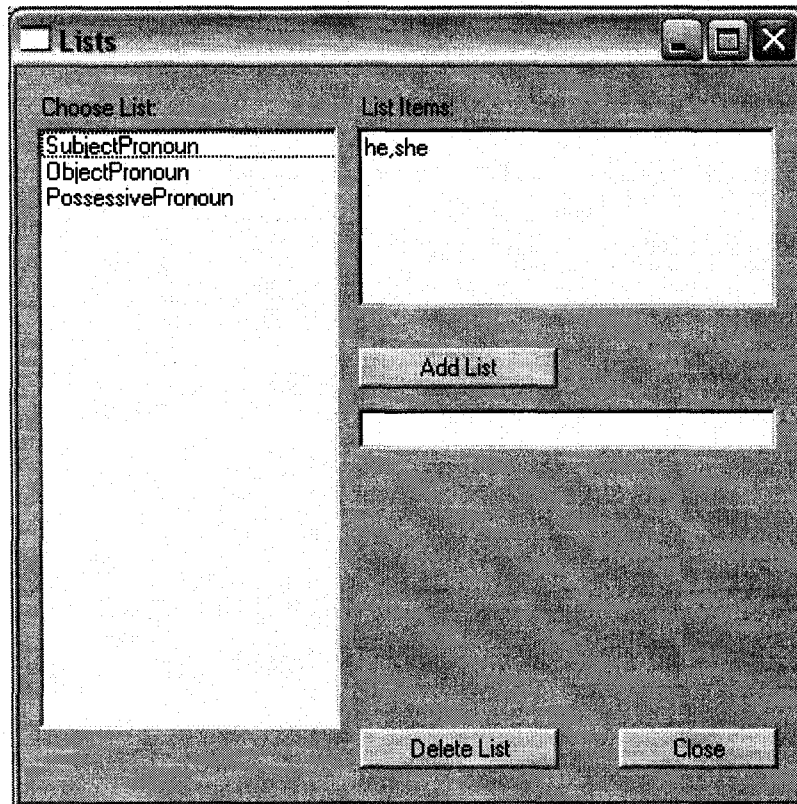
Tags Off. When you choose this option, no tags are shown.

Tags On. Choosing this option turns on the tags. When tags are turned on, if you point at any node with the mouse cursor, a tag will appear showing the name of the node. Tags are particularly useful when the Zoom is set high enough so that the text in the nodes is omitted.

Tools

Edit Lists. Choosing this option opens the "Lists" dialog box. In this box you can define a list. When a list name is used in a question, a rule is created in the grammar file. The system will accept any item on the list as a replacement for the list name. This dialog box is shown in figure 14.

Figure 14. Lists Dialog Box



For example, the sample list SubjectPronoun has two items: he and she.

So, the question Does SubjectPronoun have a valid license?

will respond to

Does he have a valid license? or Does she have a valid license?

(The above question notation " Does SubjectPronoun have a valid license?" will not actually work in Ted. The proper notation for questions is described in the next section. For now we will use the simplified notation to explain lists, although this question would properly be written "does <0 SubjectPronoun> have a valid license".)

A list can be defined to fit synonyms for a common word or phrase. For example, the list Subject has the items: he, she, the driver, the operator, the person, the owner. Now the question, "Does Subject have a valid license?", will fit six spoken questions. This is another way you can use Ted to make your system more specific to your environment.

To create a new list, type the list name (which can not contain spaces) into the "Add List" textbox. Then click the "Add List" button. The new list name will be added to the group of existing lists for the current Ted file. The new list must have a name different from that of any other list, and it must not be the same as the first name of any node. (first node names are essentially list names.)

To edit an existing list, select the list name from the "Choose List" box by clicking on it. Once a list is selected, its items can altered in the "List Items" box. Spaces are allowed within list items. The items should be separated with commas. Spaces are not required.

Another use of lists is dialect compensation. If users tend to sometimes use contractions, abbreviations, nicknames, etc. you can define a list to catch all possibilities. For example, consider the list Whats. It has two items: whats and what is. So, for questions where the phrase "what is" occur, the Whats list will match "what's" or "what is".

Lists have another very useful function that node names do not. If a list name is used in a question, not only will the system accept any item, it will remember which item was spoken. As the answer is formed at run time, it can be influenced by the spoken item. For example, let's create the list ConvType

and include the items speeding, uncovered load, driving while intoxicated, non inspection, unregistered vehicle, operating without a license, non moving violations, equipment. Now, if we include the question,

Whats Subject got for ConvType convictions?

Not only does it represent 96 spoken questions, but the answer can vary depending upon which item was spoken for ConvType. Again, the details pertaining to writing questions and answers are discussed in the next section.

To delete a list, select the list in the "Choose List" box by clicking on it. Then click the "Delete List" button.

Generate Grammar File. Choosing this option generates a full question recognition grammar file that is used by Fred, the run time component of the SAM Q/A system. This file is used to tell the system which words, phrases, and questions to recognize. The file also contains information about the answers. Once the file is generated, a "Save As" dialog box will pop up. The name of the file must be grammar.txt unless the file Fred.CPP is modified. The generation of these grammar files is the main purpose of the Ted application.

Generate Vector File. Choosing this option generates a component recognition grammar file that is used by Fred, the run time component of the SAM Q/A system. This file is used to tell the system which words and phrases to recognize. The file also contains information about the answers. Once the file is generated, a "Save As" dialog box will pop up. The name of the file must be grammar.txt unless the file Fred.CPP is modified. The generation of these grammar files is the main purpose of the Ted application.

Generate Question File. Choosing this option generates a file that includes every question and answer in the current Ted project. This is useful for system development and fine-tuning. Once the file is generated, a "Save As" dialog box will pop up. You can give the file any name you like, but the extension should be .txt since it will be a text file.

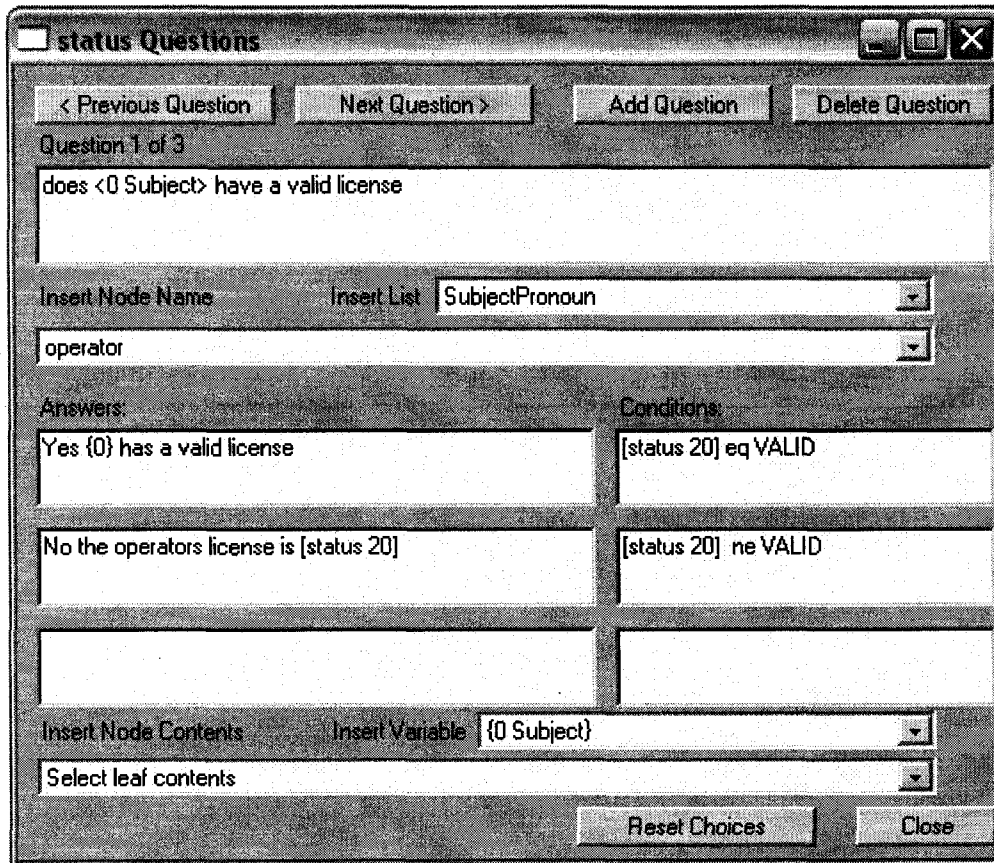
Generate Leaf File. Choosing this option generates a file that lists every leaf node in the current Ted project. The list includes the full node path and unique node number of each leaf node. This information is essential when writing the record parsing part of Fred. Once the file is generated, a "Save As" dialog box will pop up. You can give the file any name you like, but the extension should be .txt since it will be a text file.

The Node Questions Dialog Box

The Sample Questions button in any node Properties dialog box opens the node Questions dialog box. This box allows you to enter sample questions that will be associated with that node. These questions are used by Ted to generate the grammar file. The Questions dialog box is shown in figure 15.

Each node may contain any number of questions. However, you can only see one question at a time. The < Previous Question, and Next Question > buttons are used to switch between questions within the node. The current question number is shown beneath the < Previous Question button.

Figure 15. Node Questions Dialog Box



There are also buttons to add or delete questions. Most importantly, there is a text box to put the question in.

Questions

A question may contain the following elements, separated by spaces:

- Word
- Node Name
- List Name

Node Name

To insert a node name, choose from the Insert Node Name drop down list. When you insert a node name, any synonym for that node (as set in the node Properties dialog box), will be accepted in that position in the question. Note that the drop down list shows each node in the tree in a complete path. This is to help you find the node you are looking for. Once you select one, the node name will be inserted in the current cursor position. The name will be enclosed in <angle brackets>. See Reset Choices below.

List Name

To enter a list name, choose from the Insert List drop down list. Like a node name, any item in a list will be accepted. In addition, as list items are used at run time, they can be accessed by the answer in the form of a variable. See Answers below. When you choose a list name from the drop down, the list name will be inserted in <angle brackets> along with a number that Ted generates. This number is associated with the variables described below in the Answers section.

In addition to the lists you create (see the Tools menu), Ted includes several default lists. There are two types of default lists: editable, and not editable. The editable ones are SubjectPronoun, ObjectPronoun, and PossessivePronoun. These lists appear in the Lists dialog box. They can be edited or deleted, and serve as examples. However, if you look in the "Insert List" drop down box, you will notice two additional lists: SingleDigit, and

DoubleDigit. These can be neither edited nor deleted. They are more complicated than normal lists. When the grammar file is generated, these lists are handled specially. For example, if the grammar recognizes "thirty five", it will pass on the string "35". See Reset Choices below.

Question Examples

1. what state is the driver from
2. what is the drivers <dob>
3. does <0 Subject> have a valid license
4. has <0 Subject> had any tickets in the last <1 SingleDigit> years

In the second example, the node name <dob> stands for D O B, date of birth, birthday, or birth date. In the third example, the list Subject contains he, she, the driver, the operator, and the person. The fourth example uses the list SingleDigit, which can stand for for zero, one, two, ... up to nine. The number can be used as a variable in the answers and conditions, as is covered below.

Boolean Expressions

Ted uses a specific format for Boolean expressions. Since they are used in several places, they are discussed here. A Boolean expression (as far as Ted is concerned) may contain the following elements, separated by spaces:

- Word
- Node Contents
- Variable
- Operator

Node Contents

To insert the contents of a leaf node, choose a node from the Insert Node Contents drop down box. Nodes chosen in a simple Boolean expression are expected to be non-enumerated. That is, there should be only one piece of data associated with the node. If there are multiple pieces of information associated with a node, Ted will simply choose the first. This is not a limitation. There should never be a need to use enumerated leaf node data within a simple Boolean expression. This is not the case, however, when a Boolean expression is used within a filter. Filters are described below in the Answers section.

When you insert the contents of a leaf node in a Boolean expression, the contents (perhaps as part of a mathematical expression) will be compared to something at run time. If the comparison is true, the Boolean expression is true. For example, in the status Questions dialog box shown above, a Boolean expression is used as a condition (more on conditions below).

[status 20] eq VALID

The node name is "status", and it apparently is node 20. At run time, if the contents of node 20 are equal to the string "VALID", the above Boolean expression will be true. If the contents of node 20 do not exactly equal the string "VALID", the above Boolean expression will be false. See Reset Choices below.

Variable

Above, you saw that a question might include a list name chosen from the Insert List drop down box. In a Boolean expression, you may include any list used in the question. At run time, the actual list item spoken will be used in a comparison. For example, if we want to know if the driver has more than some number of points, we might use the following question.

does the driver have more than <0 SingleDigit> points

Then, in a Boolean expression, we can compare the number in the points node to the number spoken. In this case, we want to trigger an answer if the number of points is more than the number spoken.

[cpoints 35] > {0}

This Boolean expression is true if the number in node 35 is greater than the variable {0}, which is the spoken word from the SingleDigit list.

Of course, Boolean expressions using node contents might compare numbers or strings. Likewise, Boolean expressions using variables might compare numbers or strings.

See Reset Choices below.

Operator

Boolean expressions may contain a number of operators. To be specific; *, /, +, -, eq, ne, =, !=, >, <, >=, <=, &, and |.

The operators are resolved in the order shown above. Operators must have spaces to either side.

Arithmetic Operators

* Multiplication

/ Division

+ Addition

- Subtraction

You may perform arithmetic functions on numbers only (dates are numbers), not on strings. Using these operators will perform the normal functions on the two neighboring words. It is assumed that these words are numbers. The operators are resolved in the order *, /, +, -. Parentheses are not allowed. This means you can not use any grouping. You must distribute groupings before typing them into Ted.

Comparative Operators

eq compares strings to see if they are equal

ne compares strings to see if they are not equal

= equal

!= not equal

> greater than

< less than

>= greater than or equal to

<= less than or equal to

Most of the operators are comparative operators. Two of them, eq and ne, are for use only with strings. If the two neighboring strings are identical, eq will result in a true. The other comparative operators are to be used with numbers, including dates. They are =, !=, >, <, >=, and <=. Since the arithmetic operators are resolved first, these comparative expressions will compare the two neighboring arithmetic expressions (where an arithmetic expression is any number of numbers connected by arithmetic operators).

Boolean Operators

& AND

| OR

There are two Boolean operators & and |. They are resolved in that order, and parentheses are not allowed. This means that Boolean expressions must be

expanded to sum-of-products form. Although there is no negation operator, by using the ne, and != operators, any Boolean expression can be represented.

All Boolean expressions eventually resolve to a true or false. They are used in two places, which are discussed later in Filters and in Conditions. Below is an example of a valid Boolean expression in Ted.

```
[fname 6] eq BARBIE | [gender 8] eq Female & [haircolor 12] eq blond & [weight 11] <= 110 & [cpoints 35] + [lyp 36] + [2yp 37] = 0
```

This somewhat silly expression resolves to true if the driver's first name is Barbie, or if all of the following are true. She is a blond female with a weight less than or equal to 110 pounds, and all of her points total zero.

Answers

The second portion of the node Questions dialog box allows you to enter answers and conditions. The answer is spoken by the run time component if the corresponding question is detected. An answer may contain the following elements, separated by spaces:

- Word
- Node Contents
- Variable
- Operator
- Filter
- Filter Number

Node Contents

As explained above in the section Boolean Expressions, a leaf may be chosen using the "Insert Node Contents" drop down box. It is assumed that the leaf is not enumerated. To use enumerated node contents, see Filters.

Variable

The variable is also explained in the section Boolean Expressions.

Suppose the following were a question.

does the <0 Subject> have more than <1 SingleDigit> points

The answer might be:

{0} has [cpoints 35] points.

Operator

There are four operators that are acceptable in an answer, and they all deal with numbers (including dates); *, /, +, -. They must have a space on either side. So, if you want to know all of the points the driver has for the past three years, the answer would look like:

{0} has [cpoints 35] + [lyp 36] + [2yp 37] points.

Filter

A filter is an element that is used to deal with nodes that contain multiple items or values. A filter always begins and ends with parentheses, and this is the only place in a Ted project where parentheses may be used. A filter represents a

list of data from a node. For example, the filter below returns the locations for all of the accidents the driver has on record since 1995.

```
([location 63]:[accddate 60] > 12/31/1995)
```

All filters have the following format. First, a node to be returned, then a colon (:), then a Boolean expression. Spaces around the colon are not necessary. For each item in the list, if the Boolean expression is true, the item is added to the sub list returned.

So, the answer to the question

where has <0 subject> had accidents since 19 95

is

```
{0} has had accidents in ([location 63]:[accddate 60] > 12/31/1995)
```

Filter Number

A filter number works the same as a filter, but instead of returning the list of qualifying items, the filter number returns the number of qualifying items. To use a filter number, just directly precede a filter with a hash, or number sign (#).

For example, to answer the question, "list all convictions", you might use

The driver has a total of #([convtype 42]:1 = 1) convictions for the following

```
([convtype 42]:1 = 1)
```

Notice that the Boolean expression $1 = 1$ is used so that all entries will be included.

Conditions

An answer will be activated if the condition next to it true. A condition is a Boolean expression. In addition to the elements normally allowed in a Boolean expression, a condition may contain a filter number. By default, a condition is considered true if it is empty. If more than one answer is given, they should have mutually exclusive conditions. Otherwise, multiple answers to a single question might result.

Reset Choices - Important

There are four drop down boxes in the node Questions dialog box. These drop down boxes have contents that vary due to changes you make inside and outside the dialog box. Click the "Reset Choices" button to update the drop down boxes for that node.

Using Fred

Required Files

Fred is the runtime application component of the SAM Q/A system. For Fred.exe to run properly, it must be placed in a folder with a grammar file called "grammar.txt", and a record file called "record.txt". The grammar file is generated by the Ted application. Fred is compatible with either a full question grammar, or a component grammar.

The application can be started by double clicking on the Fred.exe file. This will open a window on the desktop. To ask a question of the system, click and hold the right mouse button inside the window. Ask the question into the attached microphone. Then, release the right mouse button.

The record file must be in the proper format. This format is associated with the way the question answering system is set up in Ted. In the Ted design, each piece of data is located in a leaf node of the tree. The leaf nodes are numbered. The leaf numbers can be obtained from Ted by selecting the Generate Leaf File menu option. For the Student system, this generates a file with the text shown in figure 16.

Figure 16. Student Leaf File

```
student/personal/name 5
student/personal/address 6
student/personal/dob 7
student/personal/ssn 8
student/academic/major 9
student/academic/minor 10
student/academic/advisor 11
student/academic/class 12
student/academic/status 13
student/academic/credits 19
student/academic/gpa 20
student/courses/title 14
student/courses/department 15
student/courses/number 16
student/courses/grade 17
student/courses/ech 18
```

As can be seen in the file, the name is stored in node 5. The address is stored in node 6. Some pieces of data, like course title (node 14), may have multiple items or values. An item number is used to differentiate between these multiple items. If a node contains only one item, its item number is 0. Each line in the data record must have three things: the node number where the information is stored, the item number for the data, and the data. The three pieces of information are each enclosed in square brackets.

[5][0][Jennifer Allen]

[6][0][402 south main street bivington NH]

[14][0][Introduction to Biology I]

[14][1][Chemistry I]

As long as the record is in this format, Fred will read the data when launched.

Setting the Maximum Array Sizes

There are constraints on the sizes of Fred's data structures. If an intended application will exceed these constraints, the application must be edited and recompiled. For most applications, the default values should be sufficient. However, if the record is particularly large, or has large pieces of data, Fred may need adjustment. The following constants are defined in Fred.

- LEAFS - The highest numbered leaf. This is given in the TED leaf list file.
The default value is 100 leaves.
- RECORDLENGTH - The maximum amount of characters in a record. The default value is 5000 characters.
- DUPLICATES - The maximum number of duplicate leaves in any record.
This is the number of items that one leaf might contain when a leaf contains multiple items. The default is 15.
- FIELD SIZE - The maximum number of characters in any record field. The default value is 50 characters.
- QUESTIONS - The number of template questions defined in the TED file.
The default value is 100 questions.
- QUESTIONSIZE - The maximum number of characters that will appear in any question, condition, or answer. The default value is 300 characters.

How Fred Works

Fred is the runtime portion of the SAM Q/A System. The code is written in C, and it runs on the Microsoft Windows™ operating system. Fred provides the following functionality:

- Fred reads the record file in his home directory that is called "record.txt". It parses the file, and stores the information.
- Fred connects to the speech recognition and generation engines to enable speech input and speech output.
- Fred tells the speech recognition engine to use the file "grammar.txt" as a grammar file.
- Fred reads the file "grammar.txt", which contains the question information. This file is generated by Ted. Fred stores the information including all questions, answers and conditions.
- Fred receives each spoken input as it is translated by the speech recognition engine.
- Fred formulates an appropriate response.
- Fred sends this response to the speech generation engine.

The speech recognition is performed using the Microsoft English Recognizer v5.1 recognition engine. Fred connects to this engine using the Microsoft Speech Applications Programming Interface (SAPI). Throughout this document, the phrases "SAPI", and "the SAPI speech recognition engine" are used to refer both to the speech engines themselves, as well as the connection interface.

The Code

Fred contains the following functions:

Set-up

WinMain

WndProc

ABOUTCPPMsgProc

nCwRegisterClasses

CwUnRegisterClasses

File parsing and storage

parseRecord

loadDR

parseQuestions

parseTypes

Utility functions

add

sub

mul

div

speakString

datein

dateout

substring

Response formulation

respond

resAnswer

resFilter

resBoolean

parseWordList

parseLists

replaceList

getQnum

The remainder of this chapter explains what each of these functions do, and how they interact to form a spoken question answering system.

Set-up

WinMain

This is the standard Windows window function. It creates a window and sits in a message loop until the application exits. WinMain also connects to SAPI to enable speech I/O, sets the SAPI grammar file, and opens files to be parsed.

WndProc

Again, this is a standard Windows function. It handles messages dispatched by the message loop in WinMain. This function handles mouse messages. On a WM_RBUTTONDOWN message (right mouse button pressed), SAPI is instructed to start listening for voice input. On a WM_RBUTTONUP message (right mouse button released), SAPI is instructed to stop listening. The WM_RBUTTONUP message also starts a timer. On a WM_TIMER message, if speech has been detected and recognized, the spoken input is sent to the response function.

ABOUTCPPMsgProc

This function handles messages from the "About" dialog box that close the box.

nCwRegisterClasses and CwUnRegisterClasses

These functions register and unregister classes that are used by the WinMain function.

File parsing and storage

parseRecord

This function is called from within WinMain. All of the data from the driver record are sent to it as a string (pointer to a char array). The parseRecord function parses the data from the string and stores it. For each piece of information, parseRecord increments a corresponding array element (leafLength[i]) to keep track of the number of entries in that field. Then the actual data are sent to the loadDR function.

loadDR

The driver record information is all stored in a three-dimensional array called drarray. As mentioned above, there is one space for each piece of information (each leaf number), and there may be repetitions. The loadDR function takes a string input, a leaf number, and a repetition number. It stores the string input in the appropriate location. If the input has been designated as a date within the Ted project file, the string is first sent to datein before storage.

parseQuestions

The file "grammar.txt" that is generated by Ted is used as the grammar file for speech recognition. This file also contains additional information that is ignored by SAPI. Included in the grammar file is a list of all questions, answers,

and conditions. The parseQuestions function stores all of this information in arrays to be used during response formulation.

parseTypes

The "grammar.txt" grammar file also contains a list stating the type of each leaf; string, number, or date. The parseTypes function reads the grammar file, and stores the type of each leaf in the leafType array.

Utilities

add, sub, mul, and div

These are basic arithmetic functions that operate on string representations of integers. They each take two string arguments, perform a mathematical function and return a string representation of the result.

speakString

The speakString function takes a string (char array) and converts it to a wide char array. Then it sends the wide char array to the function startSpeaking. Any string sent to speakString will be spoken by the computer.

datein

This function converts dates into "Ted Time", which is the number of days since January 1, 1900. The return type of this function is an int. The input

argument is a string representation of a date. It may be in any one of the four following formats.

- mmddy
- mmddyyy
- mm/dd/yy
- mm/dd/yyyy

dateout

This function takes a date in Ted time as an integer and returns a string representation of the date, such as "February 3 1995". This is called on any output that is listed as a date in the leafType array.

substring

The substring function is used as a utility throughout Fred. It takes three input parameters; a string to parse, a start tag string, and an end tag string. The function returns the text found between the two tags.

Response Formulation

Response formulation is the heart of the Fred application, and is supported by all of the other functions. There are four main functions involved in response formulation: respond, resAnswer, resFilter, and resBoolean.

The system works by using three types of defined expressions; the answer expression, the filter expression, and the Boolean expression. Each of the three "res" functions resolves a type of expression. An answer expression is the generalized form of the answer that was entered in Ted. An answer expression may contain filter expressions as long as they are not nested. A filter expression returns only the items from some leaf that fit certain criteria. For example, it may return the location of all accidents where people were injured ([Location] where [Number Injured] > 0). A filter can also return the number of items found rather than the items themselves. Either way, a filter expression always contains one Boolean expression.

A Boolean expression is an expression that can be evaluated to true or false. A Boolean expression may contain a numbered filter, but not a normal filter. A Boolean expression may only contain a numbered filter if it is not already contained in a filter itself, since filters can not be nested.

respond

The respond function parses the input string and finds the question number. Each question has three potential answers, and each answer has one condition. All conditions are Boolean expressions, so the conditions are sent to resBoolean. If the condition for an answer is true, respond sends the answer to resAnswer, which fills in any holes. When resAnswer returns the resolved answer, respond sends it to speakString. It is up to the developer to ensure that conditions are mutually exclusive if this is desired.

resAnswer

The resAnswer function resolves answer expressions. This consists of replacing word and leaf tags with actual data, processing arithmetic operators (+, -, *, /), and sending any filter expressions to resFilter. The resolved answer is returned.

resFilter

The resFilter function resolves filter expressions. Each filter expression has one primary leaf, and a Boolean expression. For each item in the primary leaf, a Boolean expression is built using data from the primary and other leaves of corresponding repetition number. Once the Boolean expression is built, it is sent to resBoolean for resolution. If resBoolean returns true, the primary leaf information is added to the list of matches, and the number of matches is incremented. Once all Boolean expressions have been evaluated, resFilter returns either the list of matching items, or the number of matching items, depending upon how it was called.

resBoolean

The resBoolean function resolves Boolean expressions. Most of the functionality comes from evaluating operators and their nearest neighbors. The resBoolean function also calls resFilter if the Boolean expression contains a

numbered filter. An int value of 0 is returned if the expression is false. An int value of 1 is returned if the expression is true.

parseWordList

This function is only used when the system is employing the component recognition approach. The grammar file contains a list of all words used in the sample questions, and gives them each a rarity value. The parseWordList function reads data from the grammar file and stores all of the words and rarity values.

parseLists

The meaning of the word "list" here is different from that in the above section. Here, a list is a "Ted list"; an item that represents multiple words like <Subject>. The SAPI speech recognition engine will search the grammar file for acceptable values for such a list name, but Fred also must know what values are acceptable for any list name for three reasons. When Fred is guessing which sample question is closest to the spoken input, it must know which list was used so it can add the appropriate weight. Also when Fred is resolving expressions, they may include list items. Fred needs to know which of the input items to use in a calculation or comparison. Finally, Sometimes the output includes one of these word variables. Again, Fred needs to know which word to use. The parseLists function reads the grammar file and stores all list names and acceptable values in an array called GrammarList.

replaceList

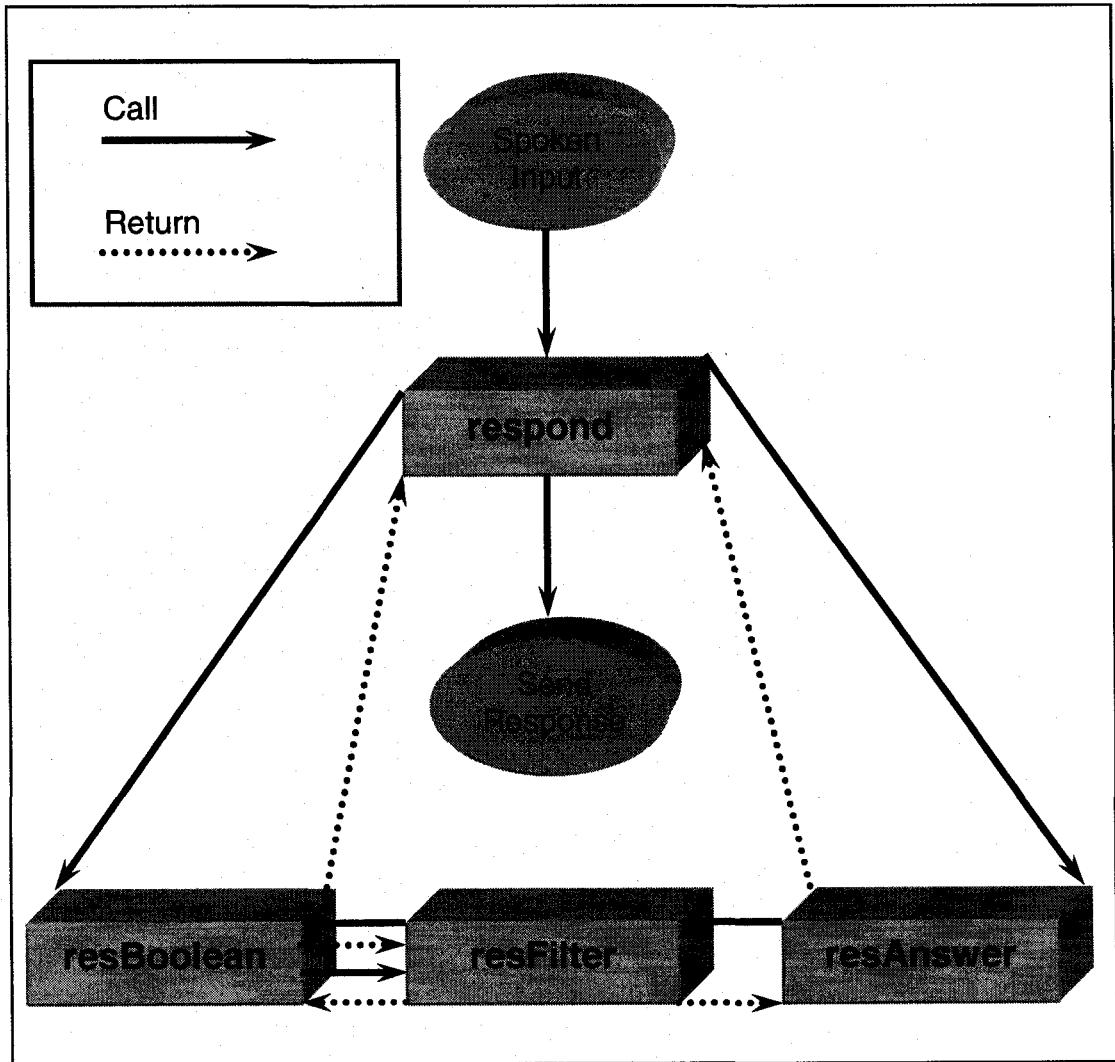
As mentioned above, there are times when Fred has an answer or condition that contains a list item. The answer or condition statement simply includes a tag number corresponding to the list name used in the question. The `replaceList` function takes this number and finds the corresponding list name in the question. Then, it finds all entries in the `GrammarList` array that use that list name. Finally, it searches the input string for a match with an acceptable value for the list name. It returns the first acceptable value it finds. Note that Fred only searches one level deep in list names.

getQnum

Fred must tell which sample question is closest to the spoken input. To do this, `respond` calls the `getQnum` function. It has access to a list of all words used in sample questions with rarity values, a list of all sample questions, and the spoken input. For each word in the list, Fred checks each sample question. If the word appears in a sample question and in the spoken input, the word's rarity value is added to the total score for that sample question. When all words have been searched, the sample question with the highest score is chosen, and the question number is returned to `respond`.

Figure 17, on the following page, shows a flow diagram depicting the response formulation process.

Figure 17. Response Formulation Flow Diagram



Flow of Control

Input goes to respond

 respond sends conditions to resBoolean

 resBoolean sends filter expressions to resFilter (numbered only)

 resFilter resolves and returns

 resBoolean resolves and returns

 respond sends answer expression to resAnswer

 resAnswer sends filter expressions to resFilter

 resFilter sends Boolean expressions to resBoolean

 resBoolean resolves and returns

 resFilter resolves and returns

 resAnswer resolves and returns

 respond sends response to speakString

Computer speaks output