

# *Saya Speech Recognition*

## *Mini Project*

Department of Computer Science

Ben-Gurion University

October 2007

Doron Meir      ID 043557214

Yaniv Zamir     ID 040871931

## Table of contents

1	Introduction.....	3
2	HMM-based Speech Recognition.....	3
3	Sphinx4 Architecture and Main Components .....	6
3.1	Sphinx4 Main Components .....	6
3.2	Sphinx4 Architecture .....	7
4	Program Design and Performance .....	9
4.1	Design.....	9
4.2	Performance.....	10
5	Conclusions.....	10
6	Future development.....	11
7	References.....	11

## Table of Figures

Figure 1: Search Graph.....	4
Figure 2 : Sphinx4 Architecture .....	7

# 1 Introduction

The goal of this project is to enhance the speech recognition of Saya, the departmental robot receptionist, using the sphinx4 recognition software currently under development at Carnegie Mellon University.

The main goal is to enable Saya to recognize speech in certain scenarios, psychologist or weatherman as an example, and in free speech mode, with good accuracy in order to enhance Saya to engage in meaningful conversations.

To do so we checked the Sphinx4 system overall performance with according to our requirements: fidelity vs. dictionary size, restricted dictionary vs. free speech and environmental considerations (microphone, background noise, etc). We wrote a dialog program for Saya using Sphinx4 for speech recognition, and a GUI program that manages the dictionaries used by our program.

In section 2 there is an overview of HMM-based speech recognition systems. In Section 3 the Sphinx4 architecture and main components are introduced. In Section 4 our programs and performance are described. In Section 5 we discuss our conclusions for the project. Finally in section 6 we discuss future development of speech recognition for Saya using the Sphinx platform.

## 2 HMM-based Speech Recognition

Sphinx-4 is an HMM-based speech recognizer. HMM stands for Hidden Markov Models, which is a type of statistical model. In HMM-based speech recognizers, each unit of sound (usually called a phoneme) is represented by a statistical model that represents the distribution of all the evidence (data) for that phoneme. This is called the acoustic model for that phoneme. When creating an acoustic model, the speech signals are first transformed into a sequence of vectors that represent certain characteristics of the signal, and the parameters of the acoustic model are then estimated using these vectors (usually called features). This process is called training the acoustic models.

During speech recognition, features are derived from the incoming speech in the same way as in the training process. The component of the recognizer that generates these features is called the front end. These live features are scored against the acoustic model.

The score obtained indicates how likely that a particular set of features (extracted from live audio) belongs to the phoneme of the corresponding acoustic model.

The process of speech recognition is to find the best possible sequence of words that will fit the given input speech. It is a search problem, and in the case of HMM-based recognizers, a graph search problem. The graph represents all possible sequences of phonemes in the entire language of the task under consideration. The graph is typically composed of the HMMs of sound units concatenated in a guided manner, as specified by the grammar of the task.

As an example, let's look at a simple search graph that decodes the words "one" and "two". It is composed of the HMMs of the sounds units of the words "one" and "two":

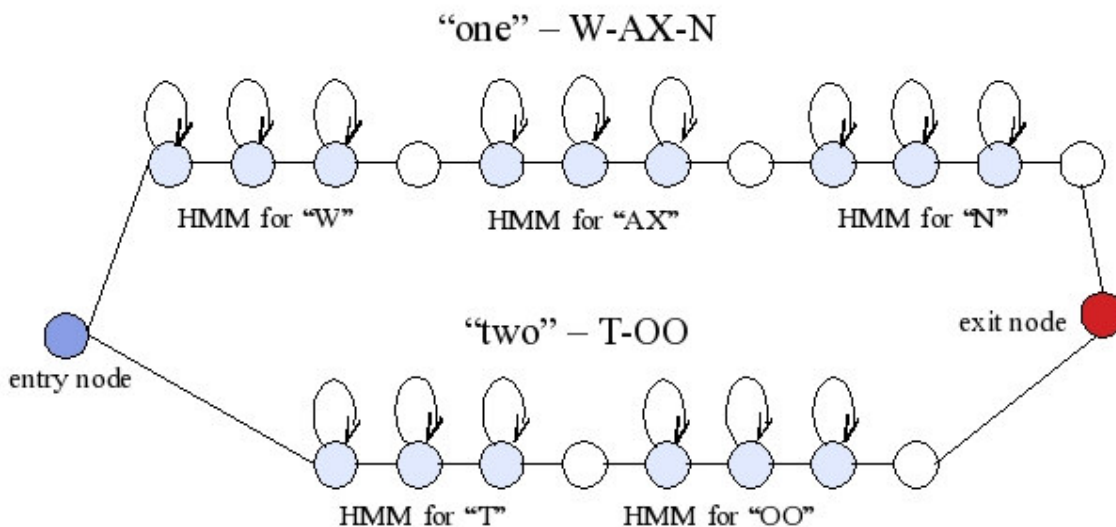


Figure 1: Search Graph

Constructing the above graph requires knowledge from various sources. It requires a dictionary, which maps the word "one" to the phonemes W, AX and N, and the word "two" to T and OO. It requires the acoustic model to obtain the HMMs for the phonemes W, AX, N, T and OO. In Sphinx-4, the task of constructing this search graph is done by the linguist.

Usually, the search graph also has information about how likely certain words will occur. This information is supplied by the language model. Suppose that, in our example, the probability of someone saying "one" (e.g., 0.8) is much higher than saying "two" (0.2). Then, in the above graph, the probability of the transition between the entry

node and the first node of the HMM for W will be 0.8, while the probability of the transition between the entry node and the first node of the HMM for T will be 0.2. The path to "one" will consequently have a higher score.

Once this graph is constructed, the sequence of parameterized speech signals (i.e., the features) is matched against different paths through the graph to find the best fit. The best fit is usually the least cost or highest scoring path, depending on the implementation. In Sphinx-4, the task of searching through the graph for the best path is done by the search manager.

As can be seen from the above graph, a lot of the nodes have self transitions. This can lead to a very large number of possible paths through the graph. As a result, finding the best possible path can take a very long time. The purpose of the pruner is to reduce the number of possible paths during the search, using heuristics like pruning away the lowest scoring paths.

As described earlier, the input speech signal is transformed into a sequence of feature vectors. After the last feature vector is decoded, we look at all the paths that have reached the final exit node (the red node). The path with the highest score is the best fit, and a result taking all the words of that path is returned.

## 3 Sphinx4 Architecture and Main Components

### 3.1 Sphinx4 Main Components

**Front End:** Accepts audio data in the form of audio files (.wav, batch files) as well as live voice from a microphone. The front end takes the input audio signal and puts it through an analysis process that includes pre-emphasis, signal segmentation, and frequency analysis.

**Linguist:** Constructs a search graph using the knowledge base. The knowledge base holds all the information for matching the incoming voice data with actual words and phrases. It is made up of three parts:

1. Acoustic Model: Holds sound data packets. Each sound packet represents a part of a sound that we make when we say a word. Each data packet represents a portion of a syllable in a word. A few data packets will make up one syllable and many data packets will eventually make up a word. This database is large because of the extent of the English language and the different portions that can make up each word.
2. Dictionary: Also called the lexicon, this contains all the words that the speech recognition engine will recognize and how they are pronounced.
3. Language Model: Contains information about probabilities of outcomes for single words and phrases. Given a set of sound data packets, it will help determine which word that it is likely to be. It also holds grammar rules so that it can determine what word will likely follow another word in a phrase. This is called an N-gram method of recognition. For example, a tri-gram would look at the previous two words of a phrase and then using probability to determine the next likely word to come.

**Decoder:** Brings together the Front-End and the Knowledge Base to output the result to the user.

## 3.2 Sphinx4 Architecture

In this section we describe the various components of Sphinx-4, and how they work together during the recognition process. First of all, let's look at the architecture diagram of Sphinx-4.

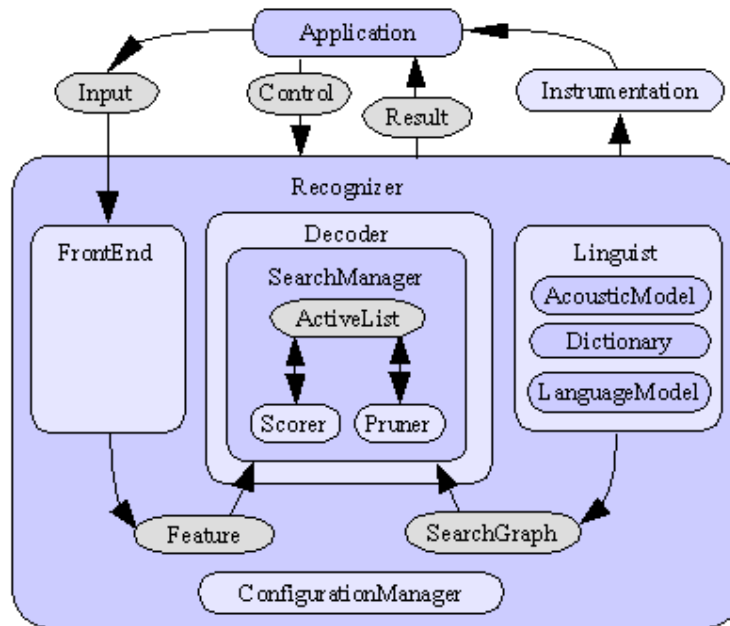


Figure 2 : Sphinx4 Architecture

The **recognizer** constructs the **front end** (which generates features from speech), the **decoder**, and the **linguist** (which generates the search graph) according to the configuration specified by the user. These components will in turn construct their own subcomponents. For example, the linguist will construct the **acoustic model**, the **dictionary**, and the **language model**. It will use the knowledge from these three components to construct a **search graph** that is appropriate for the task. The decoder will construct the **search manager**, which in turn constructs the **scorer**, the **pruner**, and the **active list**.

Most of these components represent interfaces. The search manager, linguist, acoustic model, dictionary, language model, active list, scorer, pruner, and search graph are all Java interfaces. There can be different implementations of these interfaces. The implementation to be used is specified by the user via the configuration file, an XML-based file that is loaded by the configuration manager. In the configuration file, a user can also specify the properties of the implementations. One example of a property is the sample rate of the incoming speech data.

Sphinx-4 currently implements a token-passing algorithm. Each time the search arrives at the next state in the graph, a token is created. A token points to the previous token, as well as the next state. The active list keeps track of all the current active paths through the search graph by storing the last token of each path. A token has the score of the path at that particular point in the search. To perform pruning, we simply prune the tokens in the active list.

When the application asks the recognizer to perform recognition, the search manager will ask the scorer to score each token in the active list against the next feature vector obtained from the front end. This gives a new score for each of the active paths. The pruner will then prune the tokens (i.e., active paths) using certain heuristics. Each surviving paths will then be expanded to the next states, where a new token will be created for each next state. The process repeats itself until no more feature vectors can be obtained from the front end for scoring. This usually means that there is no more input speech data. At that point, we look at all paths that have reached the final exit state, and return the highest scoring path as the result to the application.

## 4 Program Design and Performance

### 4.1 Design

The main goal of this project is to enhance Saya's speech recognition abilities, and in order to do that we wrote two applications. The dialog program and Visual Manager application. Both applications are implemented in Java, and are compatible to Sphinx4.

The dialog program is based upon the dialog demo included in the Sphinx application. Input is a word or sentence obtained through speech, in our case through a microphone. The output is a string representing the input speech. The acoustic model used by our configuration is WSJ, which is based on multiple voices reading sections of The Wall Street Journal. This provides a large vocabulary, which is needed for free speech. The program also uses tri-gram as the language model.

One can think of the program as finite automata containing states, where each state represents a mode of recognition. One such state for example could be a receptionist. For each state a different speech recognizer can be configured using a variety of dictionaries: free speech dictionary, big sentence-based dictionary, or small grammar-based dictionary. A user can switch between states either by voice command or by the given Dialog program Interface. Other features of the interface include: querying the current state, obtaining confidence scores and N-closest matching results.

The Visual Manager is a GUI program intended to simplify the use of the dialog program. A user can add, remove and update dictionaries used by the program, as well as update the main Sphinx dictionary (Lexicon). Because each dictionary works with a different recognizer, adding a dictionary involves adding a recognizer to the dialog program, which requires recompiling in order to take effect. Although updating a dictionary only requires restarting the program.

## 4.2 Performance

During the development stage, testing was performed mainly on our PC, using a standard PC microphone. Additional testing was performed on Saya.

On our PC, Shpinx4 performed poorly. For free speech, word recognition rate was very low, and for the great part the program couldn't recognize any words spoken by the user. Using grammars produced slightly better results, although still unusable practically.

The testing on Saya provided much better results. Using a sentence based dictionary containing 120 different words, the Sphinx4 platform performed well, recognizing the majority of words spoken by the user. Further testing was done using different size word dictionaries, from 200 to 2000 words. Speech recognition was less reliable as the size of the dictionary grew, when even for the 200 word dictionary the results were just adequate.

Additional testing was performed to test other system requirements: speaker recognition and recognition of words in real-time, which are both not supported yet by the Sphinx4 platform.

## 5 Conclusions

Using the Sphinx4, we were able to greatly improve the speech recognition capabilities of Saya. The Dialog program's design allows us to configure Saya's speech recognition to different states, using different dictionaries for each state. This allowed us to use specific dictionaries for specific scenarios, thus increasing the overall success of the speech recognition process. A dictionary with about 120-200 words proved to be practical for each state. For a greater amount of words the speech recognition accuracy declined.

The Sphinx4 platform proved very to be reliable, relatively easy to use, well documented and highly portable. It performs pretty well, although environmental considerations should be taken into account. For example, the difference between the performance of Spinx4 on our PC and on Saya was mainly attributed to the microphone used.

## 6 Future development

Our future plans for this project would include further customizing of the Sphinx platform in order to suite the speech recognition needs of Saya.

Firstly, thorough testing of the Dialog program, mainly comparing the performance for different dictionary types (word, sentence-based or grammar-based) and sizes.

Secondly, because of the flexibility of the Sphinx4 architecture, it supports different implementations of various components such as search manager, linguist, acoustic model, dictionary, language model, active list, scorer, pruner and search graph. Each of these components can be implemented by the user to improve system capabilities. Additionally, a speaker recognition system can also be implemented and used by the system.

Another feature that could be added is speech recognition in Hebrew, which can be done by building acoustic and language models for the Hebrew language.

## 7 References

- [1] "Sphinx4 Home," Available HTTP:  
<http://cmusphinx.sourceforge.net/sphinx4/>
- [2] "CMU Sphinx Home," Available HTTP:  
<http://www.speech.cs.cmu.edu/sphinx/>
- [3] "Sphinx-4 Whitepaper", [Online document], Available HTTP:  
<http://cmusphinx.sourceforge.net/sphinx4/#whitepaper>