Understanding the CMU Sphinx Speech Recognition System

Chun-Feng Liao
Department of Computer Science
National Chengchi University
g9104@cs.nccu.edu.tw

Abstract

The Sphinx-II is a speech recognition engine developed by CMU. It can be used to build both small, medium or large vocabulary applications. This project focused on finding out how an speech recognition engine can be implemented using HMM by referring to one of the Sphinx developer’s thesis. We dissected the source code of Sphinx-II, find out each component of an speech recognition system, then wrote some programs to evaluate the system. Because most of our research projects have been created on Java platform, we will also have some discussions about speech recognition issues on the Java platform.

1 Introduction

Voice technologies have just been put into commercial use recently because the computing resource needed by SR(Speech Recognition) and TTS(text-to-speech) is very large. In the mid- to late 1990s, personal computers started to become powerful enough to support ASR/TTS. The two key underlying technologies behind these advances are SR and TTS. SR means to transform our speech into text mode, TTS means to transform text into voice output.

The Sphinx-II is a speech recognition engine system developed at CMU. It can be used to build both small, medium or large vocabulary applications. Sphinx-II has been submitted to SourceForge as an open source project, its source code is available for download through the internet. Sphinx-II system comes from Mosur K. Ravishankar’s Ph.D thesis “Efficient Algorithms for Speech Recognition”.

This project focused on finding out how an efficient speech recognition engine can be implemented using HMM. In order to get the idea of how a it be implemented, we dissected the source code of Sphinx-II to find out the roles and functions of each
component. The detail algorithms used by this system can be found in Mosur K. Ravishankar’s thesis and it was the main reference of this project.

We wrote some demo programs to show the functionality of this recognition engine and has shown them when doing oral-presentation. The recognition rate of Sphinx speech recognition system during oral-presentation is not very good we will discuss this problem in Section 5.

2 Purposes of this Project

First we would like to find out how an efficient speech recognition engine can be implemented, we examined the source code of Sphinx2 carefully and figured out the role and function of each component. Some demo programs were given during oral presentation, the results will be discussed in this report.

Most of our previous projects were implemented on Java platform. To make this study helpful to my research domain, there is also some survey of SR(Speech Recognition) and TTS(Text to Speech) pure Java implementations include Sphinx 4 and FreeTTS in this project.

3 Basics of Speech Recognitions

3.1 Classifications of Voice Applications

There are 4 types of voice applications from easiest to hardest to implement. The first one is called IVR(Interactive Voice Response). In this sort of system, we have to communicate with computer via DTMF(Dual tone multiple frequency) then the IVR system responses us the results by playing prerecorded voice file. Most of voice application in Taiwan belong to this type. For example, the telephony course registration system, or the train-ticket ordering system.

The more advance one called Basic speech ASR, the basic speech ASR let us speak single word in order to interact with the system. Most of the commercial voice applications in USA belong to this type.
The third type is Advanced speech ASR, this sort of system can let user speak a sentence with many keyword, included some unexpected utterance (noise), like oh… uh. The SR engine behind this type of ASR can analysis the sentence we spoke than extract the information it want to know by the help of predefined “grammar”. The Grammar plays an important role in Advanced Speech ASR. The Sphinx system belongs to this type.

The last one is called “Near-natural language” ASR, which is still under research. If you talk to this type of system, you will feel as if you’re talking to a real person.

3.2 Five Basic Steps of Speech Recognition

Speech Recognition involves capturing the user’s utterance, digitizing utterance into digital signal than converting them into basic unit of utterance (phonemes), and contextually analyzing the words to ensure correct spelling for words that sound alike (such as write and right). Figure 1 illustrates these processes.

**Figure 1** Basic steps of speech recognition.

<table>
<thead>
<tr>
<th>USER</th>
<th>MICROPHONE</th>
<th>SOUND CARD</th>
<th>SPEECH RECOGNITION ENGINE</th>
<th>SPEECH-AWARE APPLICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>User speaks into the microphone</td>
<td>Microphone captures sound waves and generates electrical impulses</td>
<td>Sound card converts acoustical signal to digital signal</td>
<td>Speech recognition engine converts digital signal to phonemes, then words</td>
<td>Application processes words as text input</td>
</tr>
</tbody>
</table>

What time is it?

We can roughly divide process flow of speech recognition into five steps. They are User Input, Digitization, Phonetic breakdown, Statistical modeling, and Matching, as table 1 shows.
Table 1 Five steps of speech recognitions

<table>
<thead>
<tr>
<th>Step</th>
<th>Process Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User Input</td>
<td>The system catches user’s voice in the form of analog acoustic signal.</td>
</tr>
<tr>
<td>2</td>
<td>Digitization</td>
<td>Digitize the analog acoustic signal.</td>
</tr>
<tr>
<td>3</td>
<td>Phonetic Breakdown</td>
<td>Breaking signals into phonemes.</td>
</tr>
<tr>
<td>4</td>
<td>Statistical Modeling</td>
<td>Mapping phonemes to their phonetic representation using statistics model.</td>
</tr>
<tr>
<td>5</td>
<td>Matching</td>
<td>According to grammar, phonetic representation and Dictionary, the system returns an n-best list (i.e.: a word plus a confidence score)</td>
</tr>
</tbody>
</table>

Grammar here means the union words or phrases to constraint the range of input or output in the voice application, and Dictionary means the mapping table of phonetic representation and word, for example “the, thee” is mapping to “the”.

4 Architecture of CMU Sphinx

Sphinx2 consists of a set of libraries that include core speech recognition functions as well as auxiliary ones such as low-level audio capture. These libraries are written in C Language. Its main features includes:

- Continuous speech decoding (as opposed to isolated word recognition)
- Speaker-independent (doesn't require the user to train the system)
- Ability to provide a single best or several alternative recognitions
- Semi-continuous acoustic models
- Bi-gram or tri-gram language models

Several features specifically intended for developing real applications have been included in Sphinx2. For example, many aspects of the decoder can be reconfigured at run time. New language models can be loaded or switched dynamically. Similarly, new words and pronunciations can be added. The audio input data can be automatically logged to files for any future analysis.
4.1 Brief History of CMU Sphinx

Sphinx I was one of the world’s first user independent, high performance ASR (Automatic Speech Recognizer) developed by Dr. Kai-Fu Lee in 1987. Sphinx-I support simple word-pair grammars and generalized tri-phones.

Sphinx-II was built by Dr. Xue-dong Huang, who is currently the leader of Microsoft Speech .NET team. The Sphinx-II system was written in C and support 5-state HMM as well as N-gram language models.

Sphinx-II was said designed for speed as the Sphinx-III was designed for flexibility. Sphinx-III was developed by Eric Thayer and Mosur Ravishankar. Sphinx 4 was currently under construction, it was originally refactored from Sphinx-III system. When Sphinx 4 complete, it will be the first open source speech recognition engine fully implemented in Java Language. Sun is also one of the supporter of this project, and it will support part of Java Speech API proposed by Sun.

4.2 Components of CMU Sphinx

The high level architecture for sphinx is straightforward. As shown in the Figure 2, the architecture consists of the front end, the decoder, a knowledge base, and the application.
The front end is responsible for gathering, annotating, and processing the input data. In addition, the front end extracts features from the input data to be read by the decoder. The annotations provided by the front end include the beginning and ending of a data segment.

The knowledge base provides the information the decoder needs to do its job. This information includes the acoustic model and the language model. The knowledge base can also receive feedback from the decoder, permitting the knowledge base to dynamically modify itself based upon successive search results.

The decoder performs the main component of SR. It selects the next set of likely states, scores incoming features against these states, drops low scoring states and finally generates results.

Details of each component will be discussed in the following sections.

4.2.1 Front End

The function of Front-End API is straightforward; it provides some low-level audio access API for us to record user utterance input and play the output voice files.

4.2.2 Knowledge Base

The Knowledge base composed of three parts, they are Acoustic Model, Language Model and Dictionary.

**Acoustic models** characterize how sound changes over time. Each phoneme or speech sound is modeled by a sequence of states and signal observation probability distributions of sounds that you might hear (observe) in that state.

Sphinx2 is implemented using a 5-state phonetic model, each phone model has exactly five states. At run-time, frames of the input audio are compared to the distributions in the states to see which ones the sound could have come from which might be likely producers of the observed audio.

Acoustic models that are matched to the conditions they will be used in perform best. That is to say, English acoustic models work best for English, and telephone models work best on the telephone. With SphinxTrain, we can train acoustic models
for any language, task, or channel condition.

An LM file (often with a .lm extension) is a **Language model**. The Language model describes the likelihood, probability, or penalty taken when a sequence or collection of words is seen. Sphinx2 uses N-gram models, and usually N is 3, so they are tri-gram models, and these are sequences of three words. All the sequences of three words, two words, and one word are combined together using back-off weights in order to assign probabilities to sequences of words.

Finally, the decoder needs to know the pronunciations of words, and the **Dictionary** file (often with a .dic extension) is a list of words with a sequence of phones.

### 4.2.3 Decoder

The decoder performs main component of SR. It reads features from the front end, couples this with data from the knowledge base and feedback from the application, and performs a search to determine the most likely sequences of words that could be represented by a series of features. The term "search space" is used to describe the most likely sequences of words, and is dynamically updated by the decoder during the decoding process.

### 5 Speech in Java Platform

#### 5.1 Java Speech API

Java Speech API first released on October 26, 1998. It allows developers to incorporate speech technology into user interfaces for their Java programming language applets and applications. This API specifies a cross-platform interface to support command and control recognizers, dictation systems and speech synthesizers.

Sun has also developed the JSGF (Java Speech Grammar Format) to provide cross-platform grammar of speech recognizers. JSGF was adopt by VoiceXML 1.0 as its standard grammar. To provide cross-platform control of speech synthesizers, Sun also develop the JSML (Java Speech Markup Language).
The Java Speech API was developed by Sun Microsystems in collaboration with several famous companies such as Apple Computer, Inc., AT&T, Dragon Systems, Inc., IBM Corporation, Novell, Inc., Philips Speech Processing, and Texas Instruments Incorporated.

5.2 Implementations of Java Speech API

Although the Java Speech API was released several years ago, it seems only a few vendors are interested in this specification. I think this is due to lack of performance and low-level control of the Java platform. The following Java Speech API implementations are known to exist:

**FreeTTS and CMU Sphinx 4** are open-source implementations of Java Speech API. They will become reference implementations after the Sphinx 4 project is complete.

The FreeTTS is a speech synthesis system written entirely in the Java programming language. It is based upon Flite 1.1: a small run-time speech synthesis engine developed at CMU. Flite is derived from the Festival Speech Synthesis System from the University of Edinburgh and the FestVox project from CMU. The goal of FreeTTS is “quick and small”, because most Java applications run on the Internet. This requirement leads to a trade-off in voice quality.

**IBM Speech** for Java is the wrapper on its speech product, that is ViaVoice. So we must have ViaVoice installed on our computer before we can use this API.

**Cloud Garden** provide wrapper classes both for IBM ViaVoice and Microsoft Speech SDK.

5.3 Java Platform Issues

Developers of Sphinx 4 feel that Sphinx 4 will have less bug on memory management because GC makes managing data much easier. But on the other hand, there are several pitfalls associated with performance on the Java platform. For example, Native engines typically optimize inner loops for the CPU, but can't do that on the Java platform. Native engines arrange data to optimize cache hits and we can't really do that either.
6 Evaluating Sphinx

In order to understand the functions of the Sphinx system, we implemented several voice applications using Sphinx-II APIs. Several packages of C-based APIs help programmers to handle the record, play, and speech recognition process. We wrote 6 demo voice applications using Sphinx and FreeTTS in this project. These applications and their functions are listed below:

<table>
<thead>
<tr>
<th>Application Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphinx-II in batch mode</td>
<td>Batch mode is useful when we have a lot of pre-recorded voice files and want to extract the speech content from these files. To use this application, we have to list names of voice files in “turtle.ctl”. In this application, Sphinx system do speech recognition according to these file(turtle.ctl).</td>
</tr>
<tr>
<td>Sphinx-II in live mode</td>
<td>Live mode demonstrates an application in which the user explicitly indicates the start of each utterance using the &lt;RETURN&gt; keyboard key.</td>
</tr>
<tr>
<td>Sphinx-II in Client-Server mode</td>
<td>Runs the recognizer on a host machine, and opens a tcp socket for a single listener to connect. This returns the top N decodes. The server will block until it gets an ACK from the client. It will then start listening for one more utterance as soon as it receives the ACK. sphinx2-client automatically sends it.</td>
</tr>
<tr>
<td>A simple TTS</td>
<td>In this application you can type arbitrary</td>
</tr>
<tr>
<td><strong>Sphinx-II with FreeTTS</strong></td>
<td>English words and it will transform them into audio then play these audio out.</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>This application combines Sphinx-II with FreeTTS to show how a full-featured voice application work. You may speak any word(s) then the system will report to you what word(s) have been recognized (in speech).</td>
</tr>
</tbody>
</table>

We found that the performance of Sphinx-II is not very well, this is because the sample language model it provides was trained for American users. There is a tool called Sphinx-Train can train new language by oneself available for download on the Sphinx official site.

Another question is what should we do if we have to speak in Mandarin instead of English? The problem is the acoustic model for Sphinx is dedicated to the English, if we have to recognize Mandarin, we have to build the acoustic model again. Before building them we need to analysis the pronunciation of Mandarin then extract the phonemes. These are not easy job for us, moreover, several good recognition engine for Mandarin have been successfully developed. Some of them are commercial and some are free. Because our research will not focus on implementing SR engine, so using these mature SR engine product as basis of our research is acceptable.

We discovered that the Sphinx-II-with-FreeTTS application was very slow. Although we use multithread to prevent the I/O blocking delay, sometimes the TTS prompt was not speak even more then 5 seconds. The Sphinx-II was written in C as the FreeTTS was fully implemented in Java. To bind these two different types of system together, there will be some performance overhead. We have to unify the implementation platform to get better performance. Most of our previous projects were developed on Java platform so we choose to wait for Sphinx 4 to release. Alternative solution may be using the Java Speech API implementations provided by IBM or Cloud Garden.
7 Conclusion

In this project we found that the core of Sphinx speech recognition system is composed of three components: FrontEnd, Knowledge base and decoder. FrontEnd receives and processes speech signals. Knowledge base provides data for decoder and the decoder searches the states then return the result. A speech recognition system provides some APIs to be manipulated by voice applications.

In order to see how an full-function voice application (full-function here means using SR and TTS simultaneously) works, we use the FreeTTS as the TTS engine, and combine it with Sphinx-II. The problem is we can’t expect good performance to the cross-platform solutions, so we have to use an full java implementation like Sphinx 4.

For the reason of easy to develop and integrate with previous projects, we choose to use java platform products. But in my opinion the SR/TTS fully implemented in Java may be suitable for research and prototyping and hard for commercialize. Because there are many performance issues to overcome on this platform.

8 Reference