Researching the Sphinx library for audio transcription

ITMT 594 – Special Projects in IT: Inventing Text-Searchable Video

Brian Bailey

Summer 2013

Illinois Institute of Technology

Abstract

The idea of combining the Sphinx Open Source speech recognition toolkit with some web based editing software to allow for automated class lecture video transcription was the inspiration for this research. The Sphinx library is an open source library, written in both C and Java, that we investigated using to automate the transcription process. There are numerous settings that can be fine tuned in the library configuration file and we found the audio format and quality greatly impact the results of the transcription. This Sphinx library was used in this project we called Factor. A Java application using the Java version of the library was developed and tested along with a Python application to batch process audio transcription on a Eucalyptus cloud infrastructure.

Introduction

For this project we were investigating integrating various systems and applications for creating a video to speech transcription service/application. The goal was to research a series of applications that would be combined together to produce an integrated system for transcribing class lectures, storing the results in a database, providing an interface to view and edit the transcriptions while viewing the original video, and provide a system to allow the creation of a DocBook xml file that could be rendered out to different output formats based on a user's needs.

The first component of this research was a project we named Factor. This was the part of the project that was to deal with taking class lecture videos and transcribing them using the Carnegie Mellon University (CMU) Sphinx toolkit. The idea here was to build an automated application that takes the class lecture videos and uses the CMU Sphinx open source library to generate transcripts with timecode for our other components of the project to use as source input for their needs. This component of the entire project, named Factor, really revolved around getting the audio in the correct format from the class lecture videos and using the CMU Sphinx library to produce closest possible transcriptions. This paper deals mostly with this project.

Another component of our research was a project we called Window Pane. Window Pane would ideally be an easy to use web based application for viewing course lecture videos and transcripts side by side. In addition to viewing the transcripts there would be a way of editing the text of the transcripts so mistakes made by the Sphinx based application could be corrected as the video was watched. These changes would be saved back to the database for future users to view. Some form of gamification could be introduced to encourage users to participate.

The third component of our research was a project that involved building an application for taking the transcript data and allowing users to markup these transcripts into the DocBook xml format. This would be a web based application that would allow users to markup the raw transcripts into the DocBook format in a drag-and-drop fashion. The finished DocBook xml documents could then be used to render the text to various formats such as pdf or epub.

The final component of our research involved looking into various database systems and architectures to best allow for storage and retrieval for these various systems. We have looked into numerous systems such as relational databases like MySQL and object based no SQL databases like CouchDB. Different database systems had advantages and disadvantages for the different aspects of these projects.

The team involved in this research included:

- Brian Bailey – bbailey4@hawk.iit.edu
- Mike Fitch – jfitch@hawk.iit.edu
- Jeremy Hajek – hajek@iit.edu
- Matt Hoekstra – mhoekst1@hawk.iit.edu
- Christopher King – cking4@hawk.iit.edu
- Meghashree Ramachandra – mramacha@hawk.iit.edu

Introduction to Sphinx Toolkit

The CMUSphinx toolkit is a speech recognition toolkit that was developed by Carnegie Mellon University. The toolkit consists of numerous open source libraries and software applications. The core library being the Sphinx library itself and currently it is being developed in two primary branches. These two releases include Pocketsphinx and Sphinx4.

Pocketsphinx is a lightweight speech recognizer library that is written in C. It would generally be considered the fastest of their systems. Since it is written in pure C it works well with projects that are already using C based applications. Being a very fast speech recognition engine it works well for real-time applications, desktop and mobile applications, and command and control applications where the speed and low resource usage are required ("Versions Of Decoders", 2013).

Sphinx4 is their "state-of-the-art" recognition engine that has been written completely in Java. Since it is written in Java, it doesn't have quite the same speed or low resource usage as the pure C library does. The Sphinx4 decoder is easier to work with and configure though. Sphinx4 is a good choice for web applications and cloud computing projects ("Versions Of Decoders", 2013).

There is an older decoder called Sphinx3. This decoder is the predecessor to Sphinx4 and is also written entirely in C. CMU only provides this version for research and it should not be used in active projects. Many researchers use it to compare accuracy by using its results as a baseline for comparison. There is also an obsolete version called Sphinx2. Sphinx2 is the predecessor to Pocketsphinx ("Versions Of Decoders", 2013).

Based on the research I have done I think we should be focused on using the Sphinx4

Java library. It seems to be easier to work with quickly and also the better choice for web based projects. Pocketshpinx is the pure C library and that would be useful for integrating with all kinds of other languages and hardware devices but that is not necessarily our intended use case. On the before you start page of the wiki, it suggests Pocketshpinx for speed and portability and Sphinx4 for flexibility and manageability ("CMUSphinx Wiki", 2013).

The versions of decoders page of the CMUSphinx Wiki gives a short paragraph description of each. It states on that page that Pocketshpinx has Python integration and I see Python files included with the source. I didn't try working with Pocketshpinx or any of its Python integration during this research. The description they provide for Sphinx4 describes our use case very closely.

> "Sphinx-4 is a state-of-the-art speech recognition system written entirely in the Java™ programming language. It's best for implementation of complex server or cloud-based system with deep interaction with NLP modules, web services and cloud computing. For further detail, please check the Sphinx-4 page." ("Versions Of Decoders", 2013)

The following web pages are the best spot to start researching Sphinx:

- Sphinx General Wiki: http://cmusphinx.sourceforge.net/wiki/
- Sphinx Versions Page: http://cmusphinx.sourceforge.net/wiki/versions
- Sphinx 4 main page: http://cmusphinx.sourceforge.net/sphinx4/
- Sphinx Downloads: http://cmusphinx.sourceforge.net/wiki/download/

- Sphinx Application Programming Guide: http://cmusphinx.sourceforge.net/sphinx4/doc/ProgrammersGuide.html
- Speech at Carnegie Mellon University: http://www.speech.cs.cmu.edu/

Other Sphinx Tools

The CMUSphinx toolkit contains additional software tools beyond the Sphinx libraries. There is a Sphinxbase support library that is required when using Pocketshpinx and some of the training tools. One of these training tools is named Sphinxtrain. It is used to do acoustic model training. Another is titled CMUclmtk and it is used to work with language models ("CMUSphinx Wiki", 2013).

Another thing you need when working with the Sphinx library is language and acoustic models. The library's source download includes some basic default language and acoustic models but others are available for download on the CMU Sphinx Downloads page of the wiki.

Factor Project Development Environment

I was able to download all the source files for sphinx4 and get them open in NetBeans. I am using an Ubuntu 12.04 Desktop VM Ware virtual machine on a MAC OS X host. I installed NetBeans 7.3 using the files directly from Oracle and not the 7.1 version in the Ubuntu software store. I am using the Oracle JDK 7u21. I used the Java JDK co-bundle from Oracle that comes with NetBeans 7.3 and the JDK. This install gives you an embedded Oracle JRE as part of the JDK. This will allow you to run your apps through NetBeans and if you directly run this JRE with its full path, but since it is not on your system path by default you can't run Java apps from anywhere in the terminal. I went ahead and in the Ubuntu software center installed the OpenJDK Java 7 Runtime. This is the open source version of the Java runtime provided by the community and the only JRE in the Ubuntu software center. It is at the same version, 7u21, as the Oracle one and will stay up to date by the ubuntu package manager. This gives you a system wide JRE in your path.

I also used the Illinois Institute of Technology provided Eucalyptus cloud to do some of my testing. The Eucalyptus cloud is API compatible with Amazon web services. I deployed my Java application and Python script to a Eucalyptus cloud instance to process large batches of audio files during testing. This allowed me to process a folder of audio files with different configs and not tie up my local machine for hours.

Sphinx4 Source Code

I downloaded both the sphinx4 source and binaries. The binaries zip has all the demo applications and libraries compiled and provided ready to run and test. The source zip has all the source code files and a NetBeans project that includes all the demo applications and libraries in source form. The NetBeans project opened just fine but there were some errors from missing libraries. Numerous projects were missing the jsapi classes.

If you go to the lib directory in the source package you will see a README.txt file that describes how to fix the problem. You need to run a shell script to build the jsapi.jar file because it is distributed under a different license than the Sphinx source. All you need to do is run the jsapi.sh shell script like the readme describes and it should build the jsapi.jar for you.

The first time I tried to run it I got an error. The error was related to uudecode. The third paragraph in the readme explains this error. I went to the terminal and installed the required package using “sudo apt-get install sharutils”. I ran the shell script again and everything worked as expected.

I reopened the projects in NetBeans and after NetBeans had a chance to rescan all the projects the errors and warnings went away.

Sphinx Data Models

There are different types of data models used with sphinx. There are Language Models, Acoustic Models, and Dictionaries. Acoustic models require large amounts of recorded audio to create. There are various high-quality acoustic models available for download. Language models are models that describes a language. These are also available for download. For this project we decided to start with existing data models and work from there. Creating a new language or acoustic model would involve considerable source material and time. Based on info from the *Training Acoustic Model For CMUSphinx* page of the CMUSphinx wiki we shouldn't need to train an acoustic model for this project. We should be able to do acoustic model adaptation to improve performance and accuracy (“CMUSphinx Wiki”, 2013).

During the term of this project we did not get to the point of testing adapting one of the default data models or adding to a language model or dictionary. I will describe the results we received and some of the additional steps we could take in a subsequent section of this report.

I have provided some web page links here that are related to downloading, creating, modifying, and adapting Sphinx data models:

- Model Downloads:

  http://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language%20Models/

- Building language models:

  http://cmusphinx.sourceforge.net/wiki/tutoriallm

- CMUclmtk Development (Language Model Toolkit):

  http://cmusphinx.sourceforge.net/wiki/cmuclmtkdevelopment

  http://www.speech.cs.cmu.edu/SLM/toolkit_documentation.html

  http://www.speech.cs.cmu.edu/SLM/toolkit.html

- Sphinx lmtool (Web Based Language Model Tool) :

  http://www.speech.cs.cmu.edu/tools/lmtool.html

- Building an acoustic model:

  http://cmusphinx.sourceforge.net/wiki/tutorialam

- Adapting the default acoustic model:

  http://cmusphinx.sourceforge.net/wiki/tutorialadapt

- Building or adding to a dictionary:

  http://cmusphinx.sourceforge.net/wiki/tutorialdict

Factor Project Application Workflow

Described here is the general workflow for our Factor project. A class recording is submitted to the system for processing. First it would be converted to the format that

Sphinx needs. Various instructors may use different settings and programs for their recordings. This may mean just an audio conversion or maybe even stripping the video from the audio. Then this audio is processed through our sphinx application to produce our starter transcript. The starter transcript includes each word Sphinx detected and the start and end time of that word. At this point we would take this transcript and pass it on to the web based applications and databases that are also a part of this research.

Sphinx Preferred Audio Format

From what I see the audio needs to be encoded in a specific format based on the acoustic model but can be saved in any file format readable by Java sound. The transcriber demo says you have to use 16-bit mono signed PCM-linear, 16kHz, uncompressed, little-endian. Other parts of the documentation mention this format too.

The CMUSphix wiki page dedicated to training acoustic models describe the formats required when training or adapting an acoustic model which align with the audio format you would need to use for that acoustic model. The training recordings need to be in MS WAV format with a 16kHz, 16 bit, mono format for desktop use and 8kHz, 16 bit, mono for telephone use. It states that it is critical that the audio training be in that format ("Training Acoustic Model For CMUSphinx", 2013). I found the sphinx library will error during runtime if the audio is not in a format that it expects and can work with. One time I accidentally saved an audio recoding in stereo format instead of mono and my application quit with an error stating the audio was in an incorrect format. Every example application I looked at used 16kHz audio and models. For this project we focused all our attention on 16kHz, 16 bit, mono wav files for our audio format and use acoustic models created for that format.

Factor Project Applications

I was able to build a NetBeans project that takes in an audio file and does a transcription to a text file. By looking at the example project provided to us from Shafi, I was able to make some config file adjustments to adapt my application from using the standard WSJ dictionary and models included with the Sphinx library and now it is using the HUB4 dictionary, acoustic model, and language model. This should expand the word list to around 60,000 words. HUB4 is a large trigram language model. It is based off broadcast television sources and should contain about the largest vocabulary to start from of all the available models.

The full source is posted to the IIT git repository bakery. The main Factor.java source that makes up the bulk of the code for the Factor application is provided here in Appendix A. In the application, I provided multiple different configuration files. One of the configurations is based on the digits example using a fixed word list that transcribes only numbers. The configuration file named render.config.xml is the exact settings from the sample application provided by Shafi. The file factor.config.xml is the first file I started using to try to adjust different settings. I have been comparing the results from using the render config to the various factor config files. There is a config file called sphinx-custom.xml that has some settings that are set pretty different from the render sample. The config files labeled factor_1 through factor_5 have their setting set at various values between the two.

In both my application and Shafi's sample application we used the getTimedBestResult method on the recognizer's result object to get the transcribed result with timecode information. I found my results would output in the same format as the sample applications provided with the Sphinx4 library. Each line should be a sentence that the recognizer detected by the silence between them, although I didn't always find it was very accurate in that determination. Each word would be followed by parentheses with the start and end time inside and each of these groupings would be space separated. There is an option that is passed to that method to determine if it outputs silence tokens when it detects silence but we elected to not use that because we did not feel as that would be beneficial to the rest of the projects in our research. The following is a sample of how the application would output transcribed audio with timecode information.

```
hello(1.39,1.48) today(2.05,2.28) is(2.28,2.83) july(2.83,3.39) ninth(3.39,3.61)
  two(3.81,4.36) thousand(4.36,5.04)
this(5.59,5.74) is(5.74,5.97) a(5.97,6.0) based(6.45,6.84) tests(6.84,7.52)
for(8.29,8.53) those(8.53,8.58) five(8.92,9.29) ninety(9.29,9.68)
  four(9.68,9.71) project(10.42,10.7)
```

In the sample application provided by Shafi I was getting different formatted output. His results would have the start timecode information in angle brackets followed by the series of transcribed words on the next line. The following is a sample of how Shafi's application output a transcription.

```
<11.36>
 my desk or the following copy of asam
<13.98>
 by me a year cinched with old schoolhere
<19.08>
 charles i think when you when i read eden together we were neither of us what
could be called popular characters you were a sarcastic observant trude called
blotted creatureand
```

After closely looking at the code for both of our applications, I determined there should be no differences in the output unless the Sphinx4 library was modified. Speaking with Shafi confirmed that he had made some modifications to the Sphinx4 library. We determined for this project we did not want to modify the source for the library at all. Even though the library is open source and we could modify it, we determined it would be best for future research and new versions to use the library as written. Also, we felt that the default way the library output the timecode info would be more beneficial for our other web based applications.

The other application I wrote for this project was a Python script that would use the Factor.jar I wrote to do batch transcriptions of folders of audio file with multiple config files. This allowed me to run a large set of audio files with many different config files on the Eucalyptus cloud instance in a batch. You need to specify the configs that are located in your Factor Java application settings folder you want to use in the Python code and then when you run the script you pass the folder of wav files you want to process with those configs as an argument. The source code for this python script is located in Appendix B.

Factor Application Results

I found the quality of the transcriptions to be kinda mixed. Some of the words are correct and others are way off. We tried various config settings and some seemed to make some words transcribe more correctly but then others would be off. I couldn't really determine any one config file that provided better results than the other. Some worked better for some audio files and others for different audio files.

I would say overall we were probably getting less than 50% accuracy in our transcripts. One of our theories was that the audio quality was making a bigger difference in our results then the config settings. Mike tried running the audio through some basic equalizer and limiter settings to try to clean up the audio. One thing we found was that the audio quality varied greatly between instructor and course videos. Some instructors use higher quality mics and others pay more or less attention to their recording settings. I feel we had mixed results here too. It didn't really seem to overly impact the results too much.

The next thing I tried was to record myself reading a couple sentences from a book directly on my MacBook using the internal mic. I recorded these sentences right to 16 bit, 16kHz, mono, wav format. This recording was very clear and had never been compressed like some of the class video audio we had been using. With my factor_3 config file, I found the transcription to be very accurate only missing on a couple words. These were by far the most accurate results I had seen. This led me to wonder if the act of taking already compressed audio from the class videos and then converting it to the 16kHz format Sphinx wanted was one of our problems.

Mike proceeded to get some of the class lectures in the raw Camtasia format from the server that was used to produce the compressed versions. He took those versions which contained their audio as wav format and converted it to the 16kHz format Sphinx wants. We ran those through the batch process to see what kind of results we would get. Doing a quick overview on the results I would say things might be a little better but it is hard to say since these class recordings were not the exact same ones we had in the previously compressed form. What would really be helpful would be to have the same exact class audio recorded and saved in all the various formats directly. Then we would have a clear a/b comparison to determine what differences conversion and compression have. The fact that my direct recordings at the required audio format provided such good results has me wondering how all this audio conversion plays into the accuracy.

Areas for Additional Research

For this to be a useful application I think we would need to find a way to generally increase the accuracy. This may involve adjusting config settings, finding the optimum way to record the audio, finding the best way to convert the audio, training/adapting the acoustic model, added words to the dictionary or all of the above.

First let's talk about the config settings. I adjusted many of the frequently adjusted settings and while I noticed some differences I didn't notice any radical change that made me feel as if that was the only issue we were having. I think we would need to continue to test some of these settings with different audio files. It may be the case that different speakers or recording conditions would need completely different settings.

Next comes audio format. I feel this is a major factor in the results we are getting. The fact that I recorded a sample in the exact format specified and had ~90% results compared to less than 50% on one of my own class lectures leads me to believe there is something with the way we are converting/recording audio. That was the same speaker, recorded on the same laptop mic and different results. I feel there may be something happening in the saving, compressing, then converting process from the normal class videos. I would suggest some more research in this area. Including recording some speaking in the proper format then converting it to a compressed format and back to the 16kHz format and see if there is any differences between them. I am not sure what the answer is here but I feel this is an area to investigate further.

As far as data models are concerned, I would try to adapt the existing acoustic model with additional data from one of the instructors and see if that makes a difference on their videos. The CMUSphinx Wiki says, “it's enough to have 5 minutes of speech to significantly improve the dictation accuracy by adaptation to the particular speaker” (“Adapting the default acoustic model”, 2013). This involves recording know transcribed phrases in separate files for each phrase, then along with the transcription and a dictionary file describing the pronunciation of all the words in those phrases, process everything with SphinxTrain. This is a somewhat involved process that needs to be done very precisely according to the directions. These directions are located on the *Adapting the default acoustic model page* of the CMUSphinx Wiki (“Adapting the default acoustic model”, 2013).

Another issue that will need to be addressed at some point is how to add words to the dictionary. I was using the HUB4 models for my research because it has a large 60,000+ dictionary and acoustic model. I figured that would cover most normal words that would come up in a lecture. We would want to add technical terms to the dictionary if possible so they would be recognized by Sphinx. I don't feel this issue was contributing to our accuracy problem though. It was not the technical terms that were the only things that Sphinx was not transcribing correctly in my experiments. I feel this would be a way to enhance our transcriptions if we could get the accuracy of normal words up to a level we felt would be effective. The *Generating a dictionary* page of the CMUSphinx Wiki describes the process (“Generating a dictionary”, 2011).

I feel it might also be useful to get someone involved in the research that understands how speech works from a scientific perspective. Knowledge of the way speech is structured may help us understand the settings in the config file a little better. The *Concepts of speech page* on the CMUSphinx Wiki does a high level overview but it is a complex topic (“Basic concepts of speech”, 2012).

Appendix A: Factor Java Source

```
package edu.iit.factor;

import edu.cmu.sphinx.frontend.util.AudioFileDataSource;
import edu.cmu.sphinx.linguist.language.grammar.TextAlignerGrammar;
import edu.cmu.sphinx.recognizer.Recognizer;
import edu.cmu.sphinx.result.Lattice;
import edu.cmu.sphinx.result.LatticeOptimizer;
import edu.cmu.sphinx.result.Result;
import edu.cmu.sphinx.util.props.ConfigurationManager;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
 *
 * @author bbailey4
 */
public class Factor {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        String audioFileName = "";
        String configFileName = "";
        URL audioFileURL = null;
        URL configFileURL = null;

        // Get audio file path from args or set static
        try {
            if (args.length > 0) {
                File audioFile = new File(args[0]);
                audioFileURL = audioFile.toURI().toURL();
                audioFileName = audioFile.getName();
            } else {
                //File audioFile = new File("data/10001-90210-01803.wav");
                //File audioFile = new File("data/PerlPopPush.wav");
                //File audioFile = new File("data/sample.wav");
                //File audioFile = new
File("data/spring2013..100..itm100_030713.zip..itm100_030713.wav");
                //File audioFile = new File("data/spring2013..461..Class2-
Jan28.wav");
```

```
                //File audioFile = new File("data/spring2013..526..526_01_14_1.wav");
                //File audioFile = new File("data/test.wav");
                File audioFile = new File("data/test2.wav");
                audioFileURL = audioFile.toURI().toURL();
                audioFileName = audioFile.getName();
            }
            System.out.println("------------------------------");
            System.out.println(" Application Source Configuration");
            System.out.println("------------------------------");
            System.out.println("Audio File Name: " + audioFileName);
            System.out.println("Audio File URL: " + audioFileURL.toString() + "\n");
        } catch (MalformedURLException ex) {
            Logger.getLogger(Factor.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Malformed URL Error");
            System.exit(1);
        }

        // Sphinx4 Lib Configuration from config.xml in args or static
        try {
            if (args.length > 1) {
                File configFile = new File(args[1]);
                configFileURL = configFile.toURI().toURL();
                configFileName = configFile.getName();
            } else {
                //File configFile = new File("settings/digits.config.xml");
                //File configFile = new File("settings/factor.config.xml");
                File configFile = new File("settings/factor_1.config.xml");
                //File configFile = new File("settings/factor_2.config.xml");
                //File configFile = new File("settings/factor_3.config.xml");
                //File configFile = new File("settings/factor_4.config.xml");
                //File configFile = new File("settings/factor_5.config.xml");
                //File configFile = new File("settings/render.config.xml");
                //File configFile = new File("settings/lattice.config.xml");
                //File configFile = new File("settings/sphinx-custom.xml");
                configFileURL = configFile.toURI().toURL();
                configFileName = configFile.getName();
            }
            System.out.println("Settings File Name: " + configFileName);
            System.out.println("Settings File URL: " + configFileURL.toString() +
"\n");
        } catch (MalformedURLException ex) {
            Logger.getLogger(Factor.class.getName()).log(Level.SEVERE, null, ex);
            System.out.println("Malformed URL Error");
            System.exit(1);
        }

        // Declare and allocate the configuration manager
        ConfigurationManager cm = new ConfigurationManager(configFileURL);

        // Declare and allocate the resource necessary for the recognizer
        Recognizer recognizer = (Recognizer) cm.lookup("recognizer");

        // Add TextAligner Grammer Listener
        //TextAlignerGrammar grammar = (TextAlignerGrammar)
```

```
cm.lookup("textAlignGrammar");
        //recognizer.addResultListener(grammar);

        // Allocate Resources for Recgonizer
        recognizer.allocate();

        // configure the audio input for the recognizer
        AudioFileDataSource dataSource = (AudioFileDataSource)
cm.lookup("audioFileDataSource");
        dataSource.setAudioFile(audioFileURL, null);

        // Output to console Audio File Info
        System.out.println("\n------------------------------");
        System.out.println(" Audio Datasource Info");
        System.out.println("------------------------------");
        System.out.println("Datasource type : " + dataSource.toString());
        System.out.println("Audio Sample Rate (Hz) : " + dataSource.getSampleRate());
        System.out.println("Data isBigEndian : " + dataSource.isBigEndian());

        // Create output File name and path
        String outputFilePath = "";
        //outputFilePath = audioFileURL.getPath() + ".txt";
        //outputFilePath = audioFileURL.getPath() + ".factor.txt";
        //outputFilePath = audioFileURL.getPath() + ".render.txt";
        //outputFilePath = audioFileURL.getPath() + ".sphinx-custom.txt";
        if (audioFileURL.getPath() != null) {
            int pos = configFileName.indexOf(".");
            outputFilePath = audioFileURL.getPath() + "." +
configFileName.substring(0, pos) + ".txt";
        } else {
            System.out.println("Audio File Path is NULL... Exiting...");
            System.exit(1);
        }

        // Output Results
        PrintWriter pw = null;
        try {
            pw = new PrintWriter(new BufferedWriter(new FileWriter(outputFilePath,
false)));

            // Loop until last utterance in the audio file has been decoded, in which
case the recognizer will return null.
            System.out.println("\n------------------------------");
            System.out.println(" Transcription Results");
            System.out.println("------------------------------");
            Result result;
            while ((result = recognizer.recognize()) != null) {
                //Lattice lattice = new Lattice(result);
                //LatticeOptimizer optimizer = new LatticeOptimizer(lattice);
                //optimizer.optimize();
                //lattice.dumpAllPaths();
                //String resultText = result.getBestResultNoFiller();
                String resultText = result.getTimedBestResult(false, true);
                System.out.println("=> getTimedBestResult :" + resultText);
                pw.println(resultText);
            }
        } catch (IOException ex) {
            Logger.getLogger(Factor.class.getName()).log(Level.SEVERE, null, ex);
        } finally {
            if (pw != null) {
                pw.close();
                System.out.println("\n--------------------------------");
                System.out.println(" Transcription Result file path");
                System.out.println("--------------------------------");
                System.out.println("Transcription Writted to: " + outputFilePath);
            }
        }

        // Deallocate recgonizer
        recognizer.deallocate();

    } // end main
} // end Factor
```

Appendix B: Python Batch Transcribe Source

```
"""
ITM 594
Python Sphinx4 Batch Java App Processing

argv is absolute path to folder of .wav files and .xml config
files or if no arg is supplied it will use the current path
"""

import glob
import os
import subprocess
import sys
import time


if (len(sys.argv) > 1):
  root_path = sys.argv[1]
else:
  root_path = '.'

file_list = glob.glob(os.path.join(root_path, '*.wav'))
config_list = ['settings/sphinx-custom.xml', 'settings/render.config.xml',
'settings/factor_1.config.xml', 'settings/factor_3.config.xml',
'settings/factor.config.xml']

f = open(os.path.join(root_path, 'results.log'), 'w')
f.write('Batch Transcription Results\n')
f.write('---------------------------\n\n')
f.close

for file_path in file_list:
  for sphinx_config in config_list:
    # Process each file with each config
    start_time = time.time()

    # Run Sphinx4 Java App Here
    subprocess.call(['java', '-jar', 'Factor.jar', file_path, sphinx_config])

    end_time = time.time()
    f = open(os.path.join(root_path, 'results.log'), 'a')
    f.write('Processed File: ' + file_path + '\n')
    f.write('Config File: ' + sphinx_config + '\n')
    f.write('Time Elapsed: ' + str(end_time - start_time) + ' seconds\n')
    f.write('\n')
    f.close
```

Useful Web Links

Bakery IIT git Factor Repository: http://bakery.sat.iit.edu/gitlist/factor-bbailey4-git/

CMUSphinx Toolkit Wiki: http://cmusphinx.sourceforge.net/wiki/

Speech at CMU: http://www.speech.cs.cmu.edu/

Sphinx Application Programming Guide:

http://cmusphinx.sourceforge.net/sphinx4/doc/ProgrammersGuide.html

Sphinx Downloads: http://cmusphinx.sourceforge.net/wiki/download/

Sphinx4 Homepage: http://cmusphinx.sourceforge.net/sphinx4/

Model Downloads:

http://sourceforge.net/projects/cmusphinx/files/Acoustic%20and%20Language
%20Models/

Building language models:

http://cmusphinx.sourceforge.net/wiki/tutoriallm

CMUclmtk Development (Language Model Toolkit):

http://cmusphinx.sourceforge.net/wiki/cmuclmtkdevelopment

http://www.speech.cs.cmu.edu/SLM/toolkit_documentation.html

http://www.speech.cs.cmu.edu/SLM/toolkit.html

Sphinx lmtool (Web Based Language Model Tool) :

http://www.speech.cs.cmu.edu/tools/lmtool.html

Building an acoustic model:

http://cmusphinx.sourceforge.net/wiki/tutorialam

Adapting the default acoustic model:

http://cmusphinx.sourceforge.net/wiki/tutorialadapt

Building or adding to a dictionary:

http://cmusphinx.sourceforge.net/wiki/tutorialdict

References

Carnegie Mellon University. (2013, April 9). *Adapting the default acoustic model*. Retrieved from http://cmusphinx.sourceforge.net/wiki/tutorialadapt

Carnegie Mellon University. (2012, July 30). *Basic concepts of speech*. Retrieved from http://cmusphinx.sourceforge.net/wiki/tutorialconcepts

Carnegie Mellon University. (2013, July 4). *CMUSphinx Wiki*. Retrieved from http://cmusphinx.sourceforge.net/wiki/

Carnegie Mellon University. (2011, December 27). *Generating a dictionary*. Retrieved from http://cmusphinx.sourceforge.net/wiki/tutorialdict

Carnegie Mellon University. (2013, July 26). *Training Acoustic Model For CMUSphinx*. Retrieved from http://cmusphinx.sourceforge.net/wiki/tutorialam

Carnegie Mellon University. (2008). *Sphinx-4 – A speech recognizer written entirely in the Java™ programming language*. Retrieved from http://cmusphinx.sourceforge.net/sphinx4/

Carnegie Mellon University. (2012, March 26). *Versions Of Decoders*. Retrieved from http://cmusphinx.sourceforge.net/wiki/versions_