English Speech to Sanskrit Speech Translator

Abhradeep Guha Thakurta[1] Ashwath Kumar.K[2] and Hashir Khan[3]
guhathakurta.abhradeep@gmail.com[1] kashwathkumar@gmail.com[2]
hash9r@gmail.com[3]

National Institute of Technology Karnataka, Surathkal, India

Abstract. This paper presents a design level detail with an implementation insight of an English to Sanskrit Speech Translator that obtains a translation accuracy roughly 80% which is at par with similar systems developed using interlingua-based approach. The system uses both an Adaptive approach and a Non-Adaptive approach to perform the process of Language translation. The adaptive tools include Hidden Markov Model and Maximum Entropy framework whereas the Non-Adaptive techniques include Paninian grammar based parse tree mapping, techniques to restructure the words in the target sentence to conform to Indian Languages sentence structure, Longest Common Sequence based syllable splitter and heuristics based Text to Speech translator. Though the adaptive techniques that have been used are readily available but the total set of Non-Adaptive techniques and the method of integrating all the components is a unique design technique framed independently by us. The observed translating time of the system is approximately linear with respect to the input sentence. Furthermore, the system is able to translate sentences spoken in normal human speed with a rationally minimal time delay. For this reason this it can be used for online speech translation for all practical purposes. Keywords: Paninian, Longest Common Sequence, Parser,Maximum Entropy, Hidden Markov Model.

1 Introduction

Sanskrit has a very strong grammar structure and is the origin of many Indian languages. This makes it an excellent networking language which would facilitate the extension of the translator to other Indian languages. A translation scheme of this sort will have limitless applications in a land of such great diversity. Not only can this be extended to other Indian languages, a similar scheme may be used to create a translator from Indian languages to English through Sanskrit. The system under consideration is essentially grounded on four basic computational models. First, the HMM (Hidden Markov Model), Second, Maximum Entropy Model, Third, Paninian Grammar and Lastly on the varga based syllable joining and smoothening. The system is segregated into three parts, namely Speech Recognizer, Text to Text Translator (English to Sanskrit), Text to Speech Converter.

The three modules are finally integrated to run in parallel (with locks being implemented into them to take care of synchronization) in order to boost up

© A. Gelbukh, Á. Kuri (Eds.)
Advances in Artificial Intelligence and Applications
Research in Computer Science 32, 2007, pp. 363–373

Received 17/06/07
Accepted 31/08/07
Final version 22/09/07

the performance. A detailed discussion on each of these sections will be dealt throughout the course of this paper.

The Online Speech recognizer uses Hidden Markov Model[1] as the Finite State Automata for recognition of the words. It essentially uses Tied State based Silence model[1] with a special short pause model also called the tee model. The signal conditioning and the signal analysis is done with the help of Triangular Filters MFCC (Mel Frequency Cepstrum Coefficient).

The English to Sanskrit Tex translator essentially banks on Maximum Entropy based English Parser[2] and Context Free Paninian grammar [3]. The parser has a parsing accuracy of 86%. The parser runs linear with respect to the sentence length. The context free grammar based Parse Tree Mapper for English to Sanskrit uses essentially the grammar foundation laid by Panini. We assumed the language to context free mainly because of the fact that English is proven to be 85% context free, so for all practical purposes we can assume a context free scenario. We have devised a unique technique to restructure the Sanskrit sentence to follow Indian Language form which essentially makes the language nice to comprehend. We have used the most prominent classes of sentences as background for designing. Though the class list used is not exhaustive, further additions can be made with ease due to the open architecture of the system.

Sanskrit is a phonographic language, so a phonological breakup of the words is a very feasible option for deriving the underlying syllables. We use our self designed algorithm for splitting up of words into their syllables. The algorithm is mainly based on Longest Sequence Match. This whole module is independently designed by us and it has accuracy which is comparable to systems like FESTIVAL and MBROLA. For all practical purposes we have used ITRANS and C9 tagset as the language representation format.

2 English Speech to English Text Conversion

The baseline system uses the HMM-LR method, which is an integration of Hidden Markov Models [1] and generalized LR parsing [1]. In this method, the LR parsing table is used to predict phoneme candidates in speech, then these phoneme candidates are verified using the HMM phoneme models. Thus, the LR parser can be regarded as a language source model for wordtphoneme prediction/generation. The following describes the basic mechanism of HMM-LR. First, the system picks up all phonemes predicted by the initial state of the LR parsing table and invokes the HMMs to verify the existence of these predicted phonemes. During this process, all possible partial parses are constructed in parallel. The HMM phoneme verifier receives a probability array which includes end point candidates and their probabilities, and updates it using an HMM probability calculation process. This probability array is attached to each partial parse. When the highest probability in the array is lower than a threshold level, the partial parse is pruned. The parsing process proceeds in this way, and stops if the parser detects an accept action in the LR parsing table. A very accurate,

efficient parsing method is achieved through the integrated processes of speech recognition and language analysis.

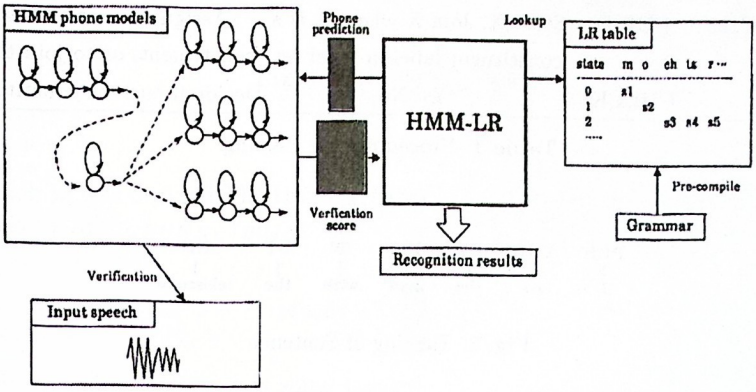

Fig. 1. HMM and LR based Automatic Speech Recognizer

3 English Text to Sanskrit Text Conversion

The English to Sanskrit Text Translator module is composed of three major sections. First, the Maximum Entropy based Parser, Second, the Parse tree Mapper between English and Sanskrit and Third, the positioning of the Sanskrit words to match Indian Language grammar form. The essence of the first two sections is inherent in the problem itself. But the third section is more of a nice to have feature than a must have feature. Sanskrit being a position independent language it is really not necessary to have a rigid positioning standard. But in most of the Indian scripts it follows a particular positioning norm which is equivalent to any Devanagari language (including Hindi). The inclusion of the positioning section is mostly to bring a smoothening effect in the language.

3.1 Maximum Entropy Parser

The parser uses four procedures, TAG, CHUNK, BUILD, and CHECK, that incrementally build parse trees with their actions. The procedures are applied in three left-to-right passes over the input sentence; the first pass applies TAG, the second pass applies CHUNK, and the third pass applies BUILD and CHECK. The passes, the procedures they apply, and the actions of the procedures are summarized in Table and described below.

First Pass The first pass takes an input sentence and uses TAG to assign each word a POS tag. The result of applying TAG to each word is shown in figure 2.

Pass	Procedure	Actions	Description
First Pass	TAG	A POS tag in tag set	Assign POS Tag to word
Second Pass	CHUNK	Start X, Join X, Other	Assign Chunk tag to POS tag and word
Third Pass	BUILD	Start X, Join X where X is a constituent label in label set	Assign current tree to start a new constituent, or to join the previous one
	CHECK	Yes, No	Decide if current constituent is complete

Table 1. Procedures for Parsing

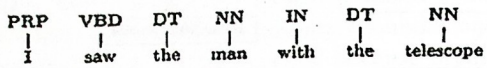

Fig. 2. Tagging of Sentence

Second Pass The second pass takes the output of the first pass and uses CHUNK to determine the "flat" phrase chunks of the sentence, where a phrase is "flat" if and only if it is a constituent whose children consist solely of POS tags.

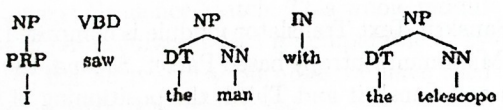

Fig. 3. After Chunking of the Sentence

Third Pass The third pass always alternates between the use of BUILD and CHECK, and completes any remaining constituent structure. BUILD decides whether a tree will start a new constituent or join the incomplete constituent immediately to its left. The result is shown in figure 4

3.2 Maximum Entropy Framework

The predicates for TAG, CHUNK, BUILD, and CHECK are used to scan the derivations of the trees in the corpus to form the training samples τ_{TAG}, τ_{CHUNK}, τ_{BUILD} and τ_{CHECK} respectively. Each training sample has the form $\tau = \{(a_l, b_1), (a_2, b_2), ..., a_N, b_N)\}$, where a_i is an action of the corresponding procedure and b_i is the list of contextual predicates that were true in the context

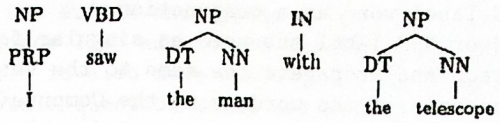

Fig. 4. After the third pass

in which al was decided.The training samples are respectively used to create the models τ_{TAG}, τ_{CHUNK}, τ_{BUILD} and τ_{CHECK} all of which have the form:

$$p(a,b) = \pi \prod_{j=1}^{k} \alpha_j^{f_j(a,b)} \tag{1}$$

where a is some action, b is some context, is a normalizing factor. a_j are the model parameters, $0 < a_j < \infty$, and $f_j(a,b)\ \epsilon\{0,1\}$ are called features, $j = 1...k$. Features encode an action $a^{'}$ as well as some contextual predicate c_p that a tree-building procedure would find useful for predicting the action $a^{'}$. feature f_j: $f_j(a,b) = 1$ if $c_p(b) =$ true && $a = a^{'}$, 0 otherwise for use in the corresponding model. Each feature f_j corresponds to a parameter a_j, which can be viewed as a "weight" that reflects the importance of the feature. For each model, the corresponding conditional probability is defined as usual

$$p(a|b) = \frac{p(a,b)}{\sum_{a' \epsilon A} p(a^{'},b)} \tag{2}$$

Based on this we use BFS(Breadth first Search on the constituent tree structures derived so far to get the parse tree with the maximum probability

3.3 Translation of Parsed English sentence to Sanskrit

Tagging based on the type In this section the words are tagged based on their part of speech. We identify case, person, number and tense of the words in this function and tag the respective words for the further translation at a later stage. This tags are stored as additional information in each node of the tree as labels. This tagging is essentially on the basis of Sanskrit grammar. The tagging proceeds based on the tags assigned at the English Parsing stage.

Algorithm: Sanskrit Tagger (root)

```
//If the tag is DT label the current word and propagate the person to
the related verb.
//If tag is WP or WRB or any other question sentence. Label the word
as third person and propagate the tag to the related verb.
//If tag is JJ label word as adjective.
```

```
//If tag is CC label word as a conjunction.
//If tag is NN or NNS label the word as singular for NN and plural for
NNS, third person and propagate the same to the verb related to the
noun. Label the case of the word using the Compute().
//If tag is VBZ or VBP then label the word (verb) as present tense.
//If tag is VBD then label the word (verb) as past tense.
//If tag is MD then label the word (verb) as future tense.
//If tag is IN then label if the word is because tag as because
//If tag is VBN or VBG then retrieve the tense label from the first
related verb.
//If tag is PRP$ then depending on word label word as first, second
or third person. Do the same for singular and plural. Also label word
as shashti (case).
//If tag is PRP extract case using Compute(), check person using CheckPerson
```

Propagate the same to the related verb. This section decides the person of the noun and the pronoun. The return values are First, Second and Third coupled with Singular or Plural.

Algorithm: Check Person(Node)

```
// Check whether the node is of the type I, we or etc If yes then take
First. If You or yours then take Second. Otherwise take Third.
// Based of the number assign add Singular or Plural to the result.
```

This section tags the Determiner(DT). This is done after all the words are tagged.

Algorithm:

```
// Check if DT is subject.
// Check if it has a related noun. If so give determiner the label the
of noun.
// Else give the determinant its own label by making it First Case.
//If DT is object perform same steps as above. Case is found by using
the Compute( Node ) in case no noun is associated.
```

The Compute() function is used to determine the case of the word based on the verb using standard Paninian Grammar rules.

Final Translation This section essentially deals with converting each word and phrase from the English sentence to its corresponding Sanskrit form following the tagging and labeling that has been done in the previous sections.

Algorithm:

```
  //With the word we find the root word in Sanskrit.
//The root word is found from a MySQL database. This also stores data
about where the suffix is stored.
//Based on suffix table and various tense, person, case, and number
```

```
labels we find the suffix and append to get the target word.
//For special words like Adjectives and Conjunction we perform a direct
translation. This is also done where the word is independent of the
labels like certain question words and determinants.
```

3.4 Restructuring of the Sanskrit sentence

The basic methodology we would be adopting is that any Indian Language is of the form <Noun Phrase> <Object> <Verb Phrase>. So the fundamentally we would be trying to restructure the Sanskrit parse tree generated by the previous section to meet this sentence form. Moreover during our course of work we have identified three major classes of sentences of English Language which have to be separately treated to achieve the goal of restructuring. In the following section we would be dealing with each of these class of sentences and their associated sub classes and conversion Heuristic is detail.

Restructuring of Verb Phrase The basic difference between the sentences in English and Indian Languages is that in English the verb is always at the beginning of the verb phrase (VP) where as in Indian Languages the object is at the beginning of the verb phrase. This is also true for Preposition Phrases (PP). An example of such a conversion is given in Figure 5.

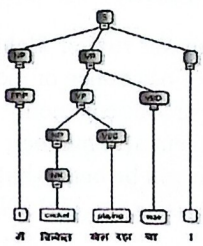

Fig. 5. Restructuring of Verb

Restructuring of WH-Phrase The class of sentences which encompass questions (essentially uses Who, What etc) are grouped together under the class of WH Phrases. In our dealing with this type of sentences we have found that the WH-Phrasecomes just before the principal verb in the Verb Phrase which it refers to. So in that case the sentences like Where were you? changes to you where were? which correctly maps on to is corresponding Hindi counterpart. We have used this technique for our restructuring.

Restructuring of No Sooner Than Sentences In computational linguistics this class of sentences is referred to as inverse sentences (SINV). This kind of sentences essentially has a to be verb (VBD) (i.e. did) and a principal verb phrase (VP). So the fundamental approach that is followed is removing all the nodes in the parse tree till the fundamental verb phrase and then placing no did just after the verb of the principal verb phrase. We have used this technique for our restructuring.

(N.B We do not claim that the classes of sentences that we have stated here is totally exhaustive. Its just a prototype and extension is always possible on similar lines.)

4 Sanskrit Text to Sanskrit Speech Conversion

Classical Sanskrit distinguishes 48 sounds. Some of these, are, however, allophones. The number of phonemes is smaller, at about 35. The sounds are traditionally listed in the order vowels (Ach), diphthongs (Hal), anusvara and visarga, stops (Spara) and nasals (starting in the back of the mouth and moving forward), and finally the liquids and fricatives. A database of the different syllables is generated. The database consists of three different recordings namely the

Vowels 15 in number
Half matras of consonants 25 in number
Semi Vowels 10 in number
Very commonly occurring syllables - 119 in number

A syllable frequency program was run for Mahabharatha and Sreebhyasa. All the syllables with a frequency of occurance of more than 1000 was recorded. It can be approximated that these are the commonly occurring syllables in Sanskrit. This number 1000 was just a heuristic number. If we increase the number of recordings, the quality of speech will be better but the space is a constraint. The labeling of the syllables is done using the ITRANS format.

4.1 Algorithm

The Procedure to be followed to convert the Sanskrit Text to Speech can be subdivided into the following steps. Since Sanskrit is a phoneme based language the conversion is easier. The input text is subdivided into smaller sentences which are again subdivided into smaller words. These words are pronounced using the pre recorded sounds of different syllables. The whole procedure can be epitomized into the following steps.

Tokenize the Sanskrit sentence to extract the words. A punctuation is used a delimiter to separate words.
Parse each word into syllables based on a syllable lookup database. The word is broken up based on the longest syllable match. The syllable database consists of the basic vowels, consonants and few commonly occurring syllables.

Join the sound files of the syllables to produce final sound of the word. Smoothening techniques are used to filter the output sound.
Join all the words with a silence (sil syllable) between them to get a continuous speech.

Tokenizing The basic idea in tokenizing is to separate a long sentence into smaller words and phrases. So every time punctuation is encountered a new token is obtained. Here Punctuations are used as delimiters to separate the sentences into words(tokens). So each of these tokens are passed on to the next stage for processing. So the Pronunciation is token by token.

Parsing During parsing each word is broken into different syllables present in the reference syllable database. The mapping of syllables is based on the largest syllable match. Initially depending on the syllables the program searches for the biggest common match between the token and the reference syllable database. During this process the program starts from the left end of the token and searches for the syllable with the largest match. Once it locates the largest match, it moves on to search the next largest syllable match in the token.

For example the word aasthaa becomes aa/s/th/aa where aa,s,th,aa are the different syllables present in the reference syllable database. `String function`

```
Parse(String s)
len←length(s);
parsed_string←NULL;
count←1;
while (count<=length)
{
//From the database find the longest syllable which is a prefix to the
string s[count  len] and store it in temp_str
parsed_string←parsed_String+temp_str+/;
count←count+length(temp_str);
}
//Remove the extra / from the end of parsed_string
return parsed_string
```

Pronunciation of each word In this module all the sound files of the corresponding syllables are added together to generate the sound file of the word. When combining the sound files of two syllables filtering techniques are used to reduce the transitions in the pronunciation of the word. To smoothen the transition between syllables, the last 100 samples of one sound is averaged with the first 100 samples of the next sound file and this replaces the 200 samples taken into consideration in the final output file. The value 100 is obtained by trial and error. The value acutually changes depending on the vargas of the two syllables.

For a better result, different values of sample merging can be used for different vargas and different combinations of the syllables.

The following plot (Figure 6) is obtained for the word haata. The plot is obtained using MATLAB. It can be observed the length of the first plot is around 12,300 samples, whereas the number of samples in the second case is reduced. Here the merging value used is 1000. In the second waveform it can be seen that theres a smoothened value near 8000 which is the transition from one syllable (haa in this case) to another syllable(tha in this case). An intermediate value is obtained near the transition which results in a smoothened sound.

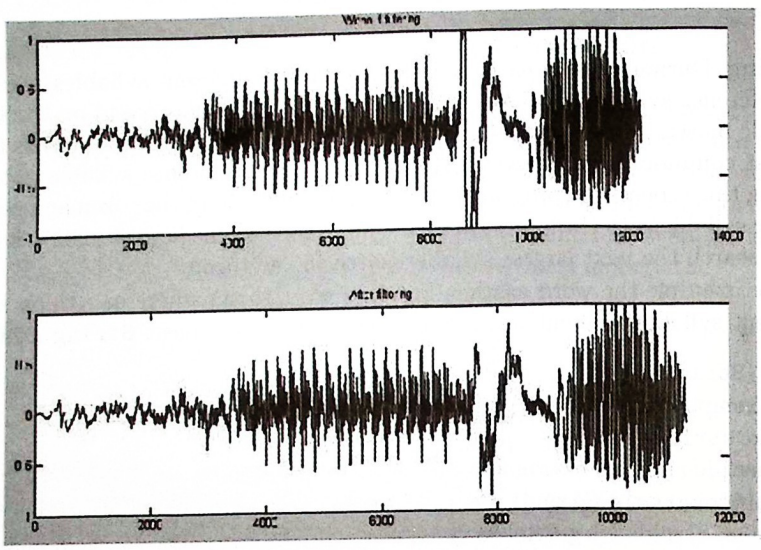

Fig. 6. Waveform for the word haata

Pronunciation of the whole sentence After each word is produced it is entered into the final sound file with a silence file in between to denote a punctuation. All the words produced are added to form a sentence by adding a pause between the words. Filtering techniques are used in this module also to reduce sharp transitions. By using the smoothening techniques the abrupt pauses are reduced and a smoothened and continuous sound can be obtained.

Filtering The filtering and smoothening of sound files while joining two sound files can be improved by using sophisticated filtering techniques. Here simple average of the two sound files has been used. Some better methods like Moving

Average Filter and Weighted filters can be used. By using complicated filtering techniques the computation time will increase affecting the overall time of execution. Even with the Simple Average Filter the overlapping sample count can be varied for different combinations of vargas. Here we have used a constant value of 100 samples for all the cases. By doing this the delivery of sound will have lesser transitions and the speech will be smoother compared to the original version.

5 Experiments

We have implemented the whole system in Microsoft Visual Studio 2003. We had tried on 35 different sentences with a word corpus of 100 words from the three different classes of sentences defined earlier. The results of the experimentation has given us a overall efficiency of 80% which is quite high keeping in mind the bottle neck situations of training the Speech Recognizer which drastically brings down the efficiency. The whole source code is available with the authors. Any one is welcome to use it.

6 Conclusion

As a part of final integration we have used a Shared Memory Model where where we run the three sections in parallel with locks implemented in them inorder to take care of empty or over flow of buffer. In a nutshell our effort has been to make a standing platform for further research in the same lines. Probably somewhere down the line a system can be made feasible which will do Real Time Emotion Based Speech Translation. If this is achieved the language barrier between Human races will be laid to rest forever. We hope for the best and encourage people to take up research work in this area to make world a better place to live in.

References

1. HMM Continious Speech Recognition using Stochastic Language Model.(Kenji KITA, Takeshi KAWABATA, Toshiyuki HANAZAWA ATR Interpreting Telephony Research Laboratories Seika-chou, Souraku-gun, Kyoto 619-02, JAPAN)
2. Linear Observed Time Statistical Parser Based on Maximum Entropy Models Adwait Ratnaparkhi, Dept. of Computer and Information Science University of Pennsylvania.
3. 3.Panini's Grammar and Computer Science Saroja Bhate and Subhash Kak,Annals of the Bhandarkar Oriental Research Institute, vol. 72,1993, pp. 79-94
4. ALTERNATIVES TO SYLLABLE-BASED ACCOUNTS OF CONSONANTAL PHONOTACTICS Donca Steriade, UCLA
5. OpenNLP resources at http://opennlp.sourceforge.net
6. Syntax-based Alignment of Multiple Translations: Extracting Paraphrases and Generating New Sentences. Bo Pang, Kevin Knight, and Daniel Marcu. Proceedings of HLT/NAACL, 2003