An Efficient Phoneme Distance Measure Using a Lexical Tree

Baruchi Har-Lev
baruchh@afeka.ac.il;

Vered Aharonson
vered@afeka.ac.il;

Ami Moyal
amim@afeka.ac.il;

ACLP – Afeka Center for Language Processing, Afeka Academic College of Engineering
218 Bney Efraim Rd. Tel Aviv 69107, Israel

Abstract – The high complexity associated with phonetic search processes when using very large lexica, long utterances and huge speech databases has become a vital issue over the last decade. In this paper, we present a novel phonetic search method based on a lexical tree that addresses this problem. The new suggested method was compared to a basic phonetic search using over 2000 utterances from the IBM Voicemail Databases and a 100k word lexicon. Preliminary results show that using a phonetic search based on the suggested tree method, does not degrade the overall recognition performance and reduces computational complexity by 35%.

I. Introduction

Phonetic search technology is commonly used in speech recognition applications, primarily for the purpose of key-word spotting [1]. The main advantage is that, once a speech database is transformed into a textual phoneme sequence, the phonetic search may be repeated many times. This is particularly useful when the list of searched words is unknown prior to the recognition process. In order to spot a key-word from a pre-defined list of words, the phonetic search engine ranks the word list according to the similarity between a word represented by its phoneme sequence and an input sequence of phonemes, and then creates a list of N-Best ranked words.

Another important challenge in speech recognition involves the use of very large vocabularies, of 100k words or more, for Large Vocabulary Continuous Speech Recognition (LVCSR) engines. A large vocabulary leads to high computational complexity and lower recognition performance. One method to address this problem is to employ a phonetic search as a pre-processing stage to the recognition engine. This enables the engine to define a list of N-Best words at the input of the LVCSR, rather than use the entire lexicon. This potentially improves recognition performance and decreases computational complexity.

However, this pre-processing stage is also highly complex when a large vocabulary is involved. The complexity stems from an exhaustive search method that is affected by several elements: the lexicon size; the length of the recognized phoneme string; the search method; and the distance measure. The goal of this research was to develop an efficient phoneme distance computation method that reduces the computational complexity of the phonetic search engine and enables a real time speech recognition operation.

Over the past few years, several studies relating to the efficiency of phonetic searches and distance measures have been conducted. Wallace, Vogt, and Sridharan [2] describes a phonetic search process combined with neural network verification. The combined process achieves better results, but focuses mainly on out-of-vocabulary terms. Hermelin, Landau and Weimann [3] describes a new efficient distance method based on data compression (Lempel-Ziv – LZ [4]), and shows that the overall complexity of the Levenshtein distance method [5] can be reduced from O (n^2) to O ($n^{1.4}N^{1.2}$) where N is the total string length. This method is mainly used for DNA queries and requires a good data compression rate, thus making it irrelevant in our case as lexicon words tend to have bad compression rates.

To this effect, we have developed a novel lexical tree based algorithm for use in phonetic search engines. Although lexical trees are generally used for storage, our suggested method makes use of the tree for phonetic searching in order to produce two main advantages. The first advantage is that, rather than checking each lexicon word against all potential input sequence hypotheses, this method enables checking each hypothesis against all possible words. Thus each hypothesis is tested only once. The second advantage is that the use of the lexical tree structure enables the grouping of several hypotheses into one hypothesis, thus allowing a collective test of several hypotheses at once. Implementation of this lexical tree mechanism enables the development of a more efficient search method.

II. Methods

A. Basic Phonetic Search

A phonetic search engine has two inputs; a lexicon (size N) of words, where each word is represented as a sequence of phonemes[1], and a sequence of recognized phonemes (length M) describing a speech utterance. The search process has two stages, where during the first stage, the search space is constructed, and in the second, the N-Best words are selected from it.

1) Search Space

We define the search space G as a grid, in which each cell contains the probability that Word(i) begins with the

[1] We used a phoneme set of 39 phonemes based on the DARPAbet

978-1-4244-8682-3/10/$26.00 ©2010 IEEE

recognized phoneme Ph(j) from the input (fig 1), thus making the size of the grid G NxM.

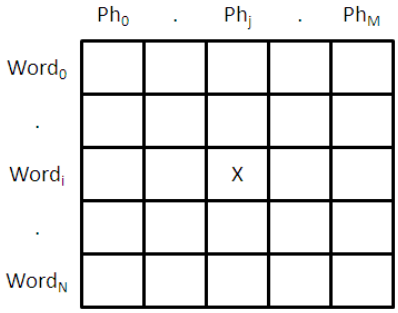

Figure 1 - Phonetic Search Grid

The value in each grid cell is computed by, $G[i,j] = Pr\{ Word\ (i)\ starts\ at\ phoneme(j)\ \}$ where $Word\ (i)$ is represented by its phonetic transcription. The probability is computed using the "Edit Distance" also known as "Levenshtein Distance" [5]. This distance metric compares two strings enabling operations of deletion, insertion, and substitution in O (n^2) complexity. In the search engine, this metric necessitates the construction of hypotheses from the recognized textual phoneme sequence in order to compute the distance.

We define a single hypothesis as a phoneme sequence of length L from an input phoneme sequence as:

$$Hyp[j,L] = \{Ph(j), Ph(j+1), \dots, Ph(j+L-1)\}$$

Where:

$$L = Length\{ Word\ (i)\}.$$

Each grid cell is thus:

$$G[i,j] = Levenshtein\{ Word\ (i), Hyp[j,L]\}$$

2) Hypothesis Stack

Each $Word\ (i)$ of length L produces $M - L + 1$ different hypotheses. We define W_{Max} as the maximum lexicon transcription length:

$$W_{Max} = \arg max\{ Length\ \{ Word\ (i)\ \}\ \}, \forall\ i \in [0, N-1]$$

And W_{Min} as the minimum lexicon transcription length:

$$W_{Min} = \arg min\{ Length\ \{ Word\ (i)\ \}\ \}, \forall\ i \in [0, N-1]\ .$$

The number of hypotheses used in the process is therefore

$$\sum_{k=W_{Min}}^{k=W_{Max}} M - k + 1$$

We define H as the hypothesis stack. Figure 2 demonstrates an example stack constructed from a sequence of 7 phonemes (M = 7) where $W_{Max} = 4$ and $W_{Min} = 1$

Sequence: {ax t aw1 t ch ey1 n}

1	ax	t	aw1	t	ch	ey1	n
2	ax t	t aw1	aw1 t	t ch	ch ey1	ey1 n	
3	ax t aw1	t aw1 t	aw1 t ch	t ch ey1	ch ey1 n		
4	ax t aw1 t	t aw1 t ch	aw1 t ch ey1	t ch ey1 n			

Sequence Transcription Length = 7
Maximum Lexicon Transcription size = 4

Figure 2 - Hypotheses stack of 7 phonemes input sequence

3) Search Space Construction Algorithm

The Algorithm for constructing the search space is as follows:

```
1   Build hypothesis stack H
2   For j = 1:M
3     For i = 1:N
4       L = Length{ Word (i) }
5       G[i, j] = Levenshtein{Word(i), Hyp[j, L]}
```

4) N-Best Selection Algorithm

Once the entire grid is computed, N-Best words should be selected via the following steps:

1 Sort the grid according to distance measure
2 Choose N words with minimum distance value
3 (Optional) Select each word once at its best value (Unique N words)

5) Complexity Analysis

The complexity of this basic phonetic search is O (Grid Construction) + O (N-Best Selection), which means that the total complexity is O (NML^2) + O (NM x Log (NM)). In this study we deal with the complexity reduction of the initial process, the search space construction. In order to reduce the complexity of this process we suggest a new method of phonetic search based on a lexical tree.

B. Phonetic Search Using a Lexical Tree

1) Tree Structure

The tree is structured as a common lexical tree: Each tree node represents a single phoneme and contains the node's level, a list of words that end with the node's phoneme and a Levenshtein matrix row of the size $W_{Max} + 1$

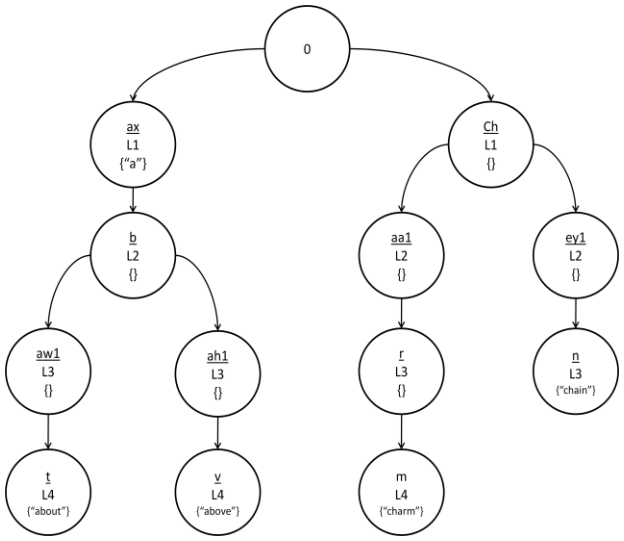

Figure 3 - 5 words lexical tree

Figure 3 demonstrates a 5 word lexicon modeled as a lexical tree.

Word	Transcription
about	ax b aw1 t
above	ax b ah1 v
a	ax
charm	ch aa1 r m
chain	ch ey1 n

Table 1 - 5 Words Lexicon

Notice, for example, that the words "about" and "a" are on the same branch, where the word "a" is represented as part of the word "about".

2) *Tree Hypothesis Stack*

One characteristic of the tree structure is that several words can be contained on one branch. This allows us to reduce the number of tested hypotheses. Thus we define the term "Contained Hypothesis" as a single hypothesis that begins with phoneme(i) of length L, and where this hypothesis is fully contained within another hypothesis that also starts at phoneme(i) but has a length of L+1. For example, the hypothesis {ax t aw1 t} has 3 contained hypotheses {ax t aw1}, {ax t} and {ax}. Using contained hypotheses enables the computation of a single Levenshtein score for several hypotheses in one pass. The total sum of hypotheses in the hypothesis stack using the tree is thus M, the length of the input phoneme sequence.

Sequence: {ax t aw1 t ch ey1 n}

1	ax	t	aw1	t	ch	ey1	n
2	ax t	t aw1	aw1 t	t ch	ch ey1	ey1 n	
3	ax t aw1	t aw1 t	aw1 t ch	t ch ey1	ch ey1 n		
4	ax t aw1 t	t aw1 t ch	aw1 t ch ey1	t ch ey1 n			

Sequence Transcription Length = 7
Maximum Lexicon Transcription size = 4

Figure 4 - Tree Hypotheses Stack

Figure 4 shows a possible tree hypothesis stack which now contains only the hypotheses shaded in gray.

3) *"Walking" The Tree*

While in a common basic phonetic search each hypothesis is fetched many times during the search process, in the tree method a single hypothesis is fetched only one time and tested against all the possible tree branches. The tree search space construction algorithm is as follows:

```
1    Create M new hypotheses in the hypotheses vector
2    for i=1:M
3       Hypothesis = fetch from hypotheses vector
4       HypLen = Length(Hypothesis)
5       for k = 1: HypLen
6          for j=1: Level Node Count(k)
7             Compute HypLen elements of the
                    Levenshtein row at Node (k, j)
8             If (Node(k, j) contain words)
9             G[Word Index, k] =   Node Levenshtein
                                   Row [node level]
```

4) *Tree Method Complexity Analysis*

The complexity of the aforementioned lexical tree method is O (Grid Construction) + O (N-Best Selection). Applying the same N-Best selection thus yields a total complexity of O (MW_{max}^2K) + O (NM x Log (NM)) where K is the average number of nodes in a tree level.

C. *Evaluation Method*

Two databases (DB) were used to evaluate the new lexical tree algorithm, IBM Voicemail I and IBM Voicemail II (VMI, II will be noted as VM) [6, 7]. The VM DB contains 2485 utterances that contain an average of 75 words per utterance and mean duration of 23 seconds. The utterances were pre-processed by a phoneme recognition engine which yielded on average strings of 260 phonemes each to process. A lexicon of 100k words was used in the process.

III. RESULTS

A preliminary computation of the algorithm coefficients in the testing DB yielded the following values $[M, N, W_{Max}, W_{Min}, K] = [260, 100k, 17, 1, 11k]$ thus making the number of hypotheses used in the tree search 260. This is approximately 1/20 of the total number of possible hypotheses. Furthermore, the number of operations needed in the tree search for creating the search space is only 0.82G, a 35% complexity reduction as opposed to a basic search which requires 1.27G (table 2).

Algorithm	Number of used hypotheses	Number of operations
Basic Search	4284	1.27G
Lexical Tree	260	0.82G

Table 2 - Theoretical Results

Table 3 shows the computation time of the two algorithms using the testing DB when 50% of the lexicon size (50k words as N, the number of selected words) is selected:

Algorithm	Grid Construction Time [Sec.]	N-Best Selection Time [Sec.]
Basic Search	38.27	0.41
Lexical Tree	39.89	0.41

Table 3 - Computation Time

Experiments were performed on a standard 2.83GHz PC.
Figure 5 shows the coverage (i.e. the number of N-Best words that appear in the reference utterance) percentage throughout the N-Best.

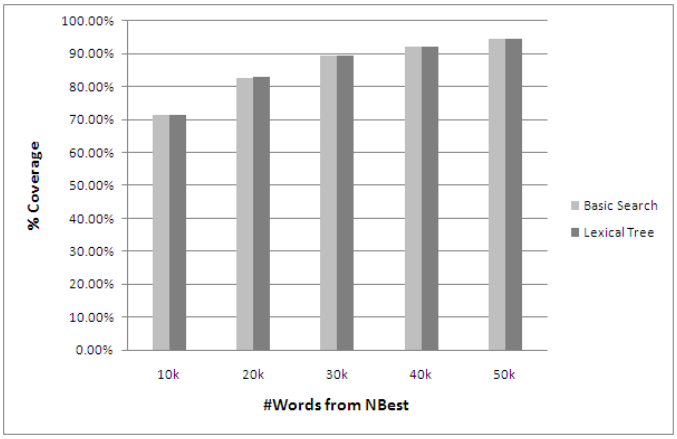

Figure 5 - Coverage in % of Lexicon

IV. Analysis and Conclusion

We presented a new phonetic search technique based on a lexical tree that differs from common basic phonetic searches in three main aspects. First, the basic search fetches each hypothesis several times during the process whereas the tree search examines each hypothesis separately and fetches it only once. A second difference is that the number of used hypotheses in the tree search is substantially smaller due to the use of "contained hypotheses" and the third difference is the ability to easily "cut" search branches off the tree, thus reducing the distance computation along the search (this can be used in future algorithms). The lexical tree method lowered the overall complexity by 35% in comparison with the common basic phonetic search. This improvement stems mainly from the use of the tree structure and of the contained hypotheses. However, when implementing and applying this novel method to the VM DB and a 100k word lexicon, we found that, although the coverage percentage remains the same throughout the N-Best, the processing time did not decrease as expected based on pre-experimental theoretical calculations. This led us to the conclusion that, although the tree structure decreases the processing time of the phonetic search by 35%, there must be an inherent limitation within the tree structure itself. The root of this limitation is likely related to the fact that tree structures are divided into two parts: the actual tree structure that contains the nodes of the tree and their arrangement; and the data contained within each node of the tree. In order to access a single Levenshtein row or the node's representing phoneme it is first necessary to access the desired node, then to penetrate the node's data, and finally, to retrieve the correct Levenshtein row or phoneme. This structure makes the tree a more complex manner of representing a word than the common basic process where a word is represented by an array of phonemes which has a simpler data structure and the Levenshtein matrix is easy to access. The following figures clearly demonstrate this problem. Figure 6 shows a Levenshtein matrix for the words "About" when comparing it to the {ax t aw1 t} hypothesis. In essence, both phoneme sequences (word and hypothesis) are represented by phoneme arrays, making it very simple to access them.

		ax	t	aw1	t
	0	1	2	3	4
ax	1				
b	2				
aw1	3				
t	4				

Figure 6 - "About" Levenshtein matrix

Figure 7 shows the same word and same hypothesis when using the tree method. The word phoneme sequence is represented as a tree, making it more difficult to gain access to each phoneme and to each Levenshtein row.

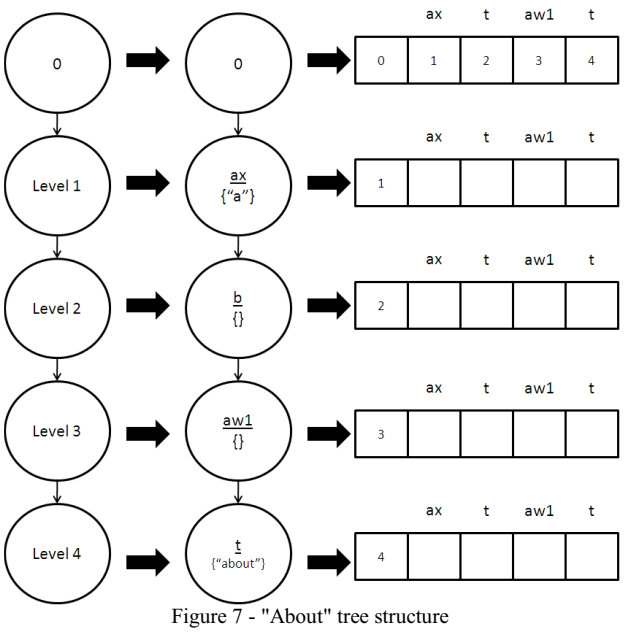

Figure 7 - "About" tree structure

Due to this fact, the tree consumes more computation time thus canceling the potential reduction from the processing time. Our assumption is that a more efficient tree structure might lead us to the desired theoretical results. Future steps towards this research will involve an improved data structure implementation for the tree-based phonetic search and extension of the mechanism to include a lattice structure at the output of the phoneme recognizer.

Acknowledgment

This research was funded by grant (#41914) provided by the Chief Scientist of the Israeli Ministry of Commerce.

References

[1] A. Amir, A. Efrat, and S. Srinivasan. 2001. Advances in Phonetic Word Spotting. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, pages 580–582, Atlanta, Georgia, USA.

[2] R. Wallace, R. Vogt, S. Sridharan. 2007. A Phonetic Search Approach to the 2006 NIST Spoken Term Detection Evaluation. In *Interspeech 2007 : 8th Annual Conference of the International Speech Communication Association*

[3] D. Hermelin, G.M. Landau, S. Landau, O. Weimann. 2009. A Unified Algorithm for Accelerating Edit Distance Computation via Text Compression. In *Proc. The 26th International Symposium on Theoretical Aspects of Computer Science*.

[4] J. Ziv, A. Lempel. 1977. A Universal Algorithm for Sequential Data Compression. In *IEEE Trans. Infrom. Theory, Vol. IT-23 pp.337-343*.

[5] M. Pucher, A. Türk, J. Ajmera, N. Fecher. 2007. Phonetic Distance Measures for Speech Recognition Vocabulary and Grammar Optimization. In *Proceedings of the Tenth International on Spoken Language Processing (Interspeech 2007 – Eurospeech),* Antwerp.

[6] B. Ramabhadran, C. Dunn, G. Ramaswamy, M. Padmanabhan, P. S. Gopalakrishnan. 1998. Voicemail Corpus Part I. *Linguistic Data Consortium (LDC)*. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC98S77

[7] B. Kingsbury, B. Ramabhadran, G. Saon, J. Huang, L. Mangu, M. Padmanabhan, S. Chen. 2002. Voicemail Corpus Part II. *Linguistic Data Consortium (LDC)*. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2002S35