Noname manuscript No.
(will be inserted by the editor)

Identifying a Basis for the Automatic Construction of Pronunciation Lexicon for Proper Names

Laxmi Narayana M · Sunil Kumar Kopparapu

Received: date / Accepted: date

Abstract Development of a proper names pronunciation lexicon is usually a manual effort which can not be avoided. Grapheme to phoneme (G2P) conversion modules, in literature, are usually rule based and work best for non-proper names in a particular language. Proper names are foreign to a G2P module. We follow an optimization approach to enable automatic construction of proper names pronunciation lexicon. The idea is to construct a small orthogonal set of words (basis) which can span the set of names in a given database. We propose two algorithms for the construction of this basis. The transcription lexicon of all the proper names in a database can be produced by the manual transcription of only the small set of basis words. We first construct a cost function and show that the minimization of the cost function results in a basis. We derive conditions for convergence of this cost function and validate them experimentally on a 163,600 large proper name database. Experiments show the transcription can be achieved by transcribing a set of 6053 basis words. The algorithms proposed are generic and independent of language; however performance is better if the proper names have same origin, namely, same language or geographical region.

Nomenclature:
TTS: Text to Speech (synthesis)
G2P: Grapheme to Phoneme (conversion)

Keywords Proper name · Lexicon · Pronunciation Dictionary · TTS · G2P · Basis · Optimization · Span deficient basis · Rank deficient basis

Laxmi Narayana M
TCS Innovation Lab - Mumbai, TCS, Yantra Park, Thane (West), Maharastra, India.
E-mail: m.laxminarayana@gmail.com

Sunil Kumar Kopparapu
TCS Innovation Lab - Mumbai, TCS, Yantra Park, Thane (West), Maharastra, India.
E-mail: sunilkumar.kopparapu@tcs.com
Tel.: +91-22-67788216

1 Introduction

Text to Speech (TTS) synthesis is an automated encoding process which converts text (a sequence of symbols conveying linguistic information), into speech (an acoustic waveform). The two major components of a TTS synthesizer are (a) natural language processing (NLP) module, which produces a phonetic transcription of the given text and (b) digital signal processing module, which transforms sequence of phones into speech [1]. Text normalization is the process of converting non-standard words like abbreviations, acronyms, dates, special symbols (for e.g. Dr, Mr, $700) into their corresponding graphemic representation [2]. Grapheme to phoneme (G2P) conversion is then performed on the normalized text. In general, an NLP module should be able to normalize the input text and map the grapheme representation of the text to a corresponding phonetic representation.

A pronunciation dictionary provides a means to map a word into its elementary phonetic components which is a key for modeling TTS synthesis systems. The reason for this is that in general, a one to one correspondence between the orthographic representation of a word and its pronunciation is absent. However, the need for a pronunciation dictionary reduces by developing a set of predefined rules (called G2P rules) developed based on linguistic knowledge, that map a sequence of characters (graphemes) into a sequence of phones. The G2P rule base is a set of rules that modify the 'default mapping' of the characters based on the 'context' in which a particular phoneme occurs. Specific contexts are matched using rules. The system triggers the rule that best fits the current context [2]. G2P converters usually produce a significant number of mistakes when converting proper names which are often of a foreign origin [3]. The rule set developed is language dependent and hence an existing rule base for one language cannot automatically be used to generate the phonetic transcription of a word from another language.

Proper names being foreign to the G2P rule base of any language, demand manual effort which is inevitable to obtain the phonetic transcription. Recently, Bonafonte et al [4] reported an average phonetic accuracy of 53% for proper names when a rule based methodology is used to construct a phonetic dictionary of proper names. Van den Heuvel et al [5] tried to automate the process of transcribing proper names by using a cascade of a general purpose G2P converter and a special purpose P2P (phoneme to phoneme) converter; the P2P converter learns from human expert knowledge. Though they report enhanced performance with the cascade system compared to direct rule based method, the performance of cascade system results in more than 30% of the name transcriptions being erroneous. In a manual effort, Font Llitjos and Black [6] adopted a web-based interface to improve pronunciation models as well as correct the pronunciations in the CMU dictionary by evaluating and collecting proper name pronunciations online. Font Llitjos and Black [7] [8] hypothesized that higher pronunciation accuracy can be achieved by adding the knowledge that people adapt their pronunciation according to where they think a proper name comes from, to a statistical model of pronunciation. The ONOMASTICA project [9] [10], a European wide research initiative, aims at the construction of multi-language pronunciation lexicon for proper names by upgrading the existing rule engines to cope with the problems posed by proper names. A significant part of the work was also devoted to the development of self-learning G2P conversion methods and the comparison of their performance with the one of rule-based methods.

A general purpose G2P rule base cannot cater to proper names because such rule bases are developed for a particular language cannot be generalized to all kinds of words, especially for the proper names. This means, there is a need to develop a pronunciation dictionary for proper names. But the development of such a lexicon[1] is not possible by having a mere rule set; it demands manual effort to generate phonetic transcriptions of a large set of names. A possible solution is to create a small set of words[2] which when phonetically transcribed, manually, can span and hence transcribe all the proper names in a given database. Obviously, the choice of the words that have to be transcribed should be such that they occur frequently in the database of names.

Several problems which were solved by constructing a cost function and finding the extremes (maxima, minima) are mentioned in literature (for e.g., [11], [12], [13]) This paper describes a method to enable construction of this set of words derived from the actual proper names database. We call it basis in a loose sense; taking cue from vector algebra. We construct a cost function which when minimized results in the identification of a basis. This can then be used in phonetic transcription of the full database of proper names. The rest of the paper is organized as follows. Section 2 formulates the identification of basis as an optimization problem. We also discuss the trivial cases of creation of a basis for proper names. Section 3 describes the proposed algorithms for basis creation of proper names. Section 4 presents experimental results and we conclude in Section 5 and also give future directions.

2 Problem Formulation

We address the following problem.

> *Given a proper name database of* $|\mathcal{N}|$ *names* (*e.g.,* {`rama, krishna, narayana ...`}), *can we construct a smaller set of words* (*basis, e.g.,* {`ra, na, krish, ya, ...`}) *automatically, such that all the* $|\mathcal{N}|$ *names can be formed by the words in the basis, namely*
>
> `rama = ra` $\oplus$ `ma`[3]
> `krishna = krish` $\oplus$ `na`
> `narayana = na` $\oplus$ `ra` $\oplus$ `ya` $\oplus$ `na`
> `...`

Given a database of proper names[4], two trivial cases of building a pronunciation lexicon are possible. (a) At one extreme one could build a pronunciation lexicon by manually transcribing all the names in the proper names database and (b) On the other extreme one could have a pronunciation dictionary of the 26 letters of the English alphabet and use that to construct the pronunciation lexicon of all the names in the dictionary by concatenating the letters that make the name. Obviously, the first trivial case is manually intensive while the second trivial case is manually easy but introduces as many joins as the number of letters that make the name; as a result the pronunciation produced is feeble. The question that one is posing here is *"Is there an optimal set of*

[1] We will use lexicon and dictionary interchangeably in this paper
[2] Words could be names themselves or part of names
[3] $\oplus$ represents a join
[4] The proper names are written in Roman script

words that one can identify and manually transcribe so that it can be used to produce a good pronunciation dictionary?". In other words, is there an optimal set of words such that the need for manual transcription is *small* and at the same time the pronunciation of the names in the database is *good?* In this paper we construct a cost function which helps us achieve a set of words (we call it the basis because it has properties of a basis) which can be used to construct the pronunciation dictionary of the full set of proper names.

We make use of a restricted definition of basis (see Appendix A) to assist our problem formulation. In our case, the *vector space* is the complete set of names in the database and the *basis* is a set of words such that, one can construct a name in the database by joining one or more words from the basis. Further, no word in the basis can be formed by joining one or more words in the basis (Property 1, Appendix A). This is analogous to the scenario of concatenative speech synthesis where one looks for the longest possible speech unit to synthesize speech with minimal discontinuities.

The optimization required is that the number of entries in the basis should be as small as possible to minimize the manual effort to transcribe them and at the same time the number of basis words (joins) used to construct a name in the database should be small. These two requirements are contradicting and hence the need for optimization.

Let $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_{|\mathcal{N}|}\}$ represent all the names in the proper names database and let $\mathcal{B} = \{b_1, b_2, ..., b_{|\mathcal{B}|}\}$ be the basis satisfying the linear independence property of Appendix A, namely for every $b_k \in \mathcal{B}; b_k$ can not be expressed as $b_i \oplus b_j \oplus ... \oplus b_l$, using any $b_i, b_j, ..., b_l \in \mathcal{B}$ and b_i or b_j or ... or $b_l \neq b_k$. This implies $b_1 \perp b_2 \perp ... \perp b_{|\mathcal{B}|}$. Additionally, for any name $\mathcal{N}_p \in \mathcal{N}$, one can write

$$\mathcal{N}_p = \oplus^n b_i \qquad b_i \in \mathcal{B} \tag{1}$$

This is equivalent to saying that $\mathcal{N}_p$ can be represented by a join of some n elements in the basis set $\mathcal{B}$ which results in $\mathcal{J}_p = (n-1)$ joins. Thus the total number of joins, $|\mathcal{J}|$ required to construct the entire database of $|\mathcal{N}|$ names is

$$|\mathcal{J}| = \sum_{p=1}^{|\mathcal{N}|} \mathcal{J}_p \tag{2}$$

2.1 Trivial cases

As mentioned earlier, two trivial cases of construction of pronunciation dictionary are possible.

Case (i): If the number of joins to construct the names is to be small then all the names in the database should be present in the basis set and this would result in the largest basis, say $\mathcal{B}_{max}$ which would have all the $|\mathcal{N}|$ names in the database. Further there is a possibility that $\mathcal{B}_{max}$ is not a basis in the sense defined in Appendix A. This is shown in Figure 1.

Case (ii): The smallest possible basis $\mathcal{B}_{min}$ would be the set of 26 letters in the English alphabet and this basis would definitely span the entire database of names, but the number of joins, $|\mathcal{J}|$ required to form the names in the database would be very large. This is shown in Figure 2.

A typical plot of the number of elements in the basis $|\mathcal{B}|$ versus the total number of joins required to construct all the names in the database $|\mathcal{J}|$, is shown in Figure 3 .

$$\begin{gathered}\mathcal{N}_1 = b_1 \\ \mathcal{N}_2 = b_2 \\ \mathcal{N}_3 = b_3 \\ . \\ . \\ . \\ \mathcal{N}_{|\mathcal{N}|} = b_{|\mathcal{N}|} \\ . \\ \mathcal{B} = \mathcal{N}; |\mathcal{B}| = |\mathcal{N}|; |\mathcal{J}| = 0\end{gathered}$$

Fig. 1 Trivial *Case (i)* where $|\mathcal{B}| = |\mathcal{N}|; |\mathcal{J}| = 0$

$$\begin{gathered}\mathcal{N}_1 = \alpha_1 \oplus \alpha_{25} \oplus \alpha_5 \\ \mathcal{N}_2 = \alpha_1 \oplus \alpha_{12} \oplus \alpha_3 \\ \mathcal{N}_3 = \alpha_{11} \oplus \alpha_{15} \oplus \alpha_{17} \oplus \alpha_{29} \\ . \\ . \\ . \\ \mathcal{N}_{|\mathcal{N}|} = \alpha_{14} \oplus \alpha_{20} \\ . \\ \mathcal{B} = \{\alpha_i\}_{i=1}^{26}; |\mathcal{B}| = 26; |\mathcal{J}| \approx \infty\end{gathered}$$

Fig. 2 Trivial *Case (ii)* where $|\mathcal{B}| = 26$ and $|\mathcal{J}| \approx \infty$

The scenario depicted in *Case (i)* corresponds to the point A in Figure 3 and the *Case (ii)* corresponds to the point B in Figure 3. We believe that the cost of construction of basis would be maximum at these two extreme trivial cases. Probably there is a case between these two trivial solutions; like the knee point C at which the cost of construction of the basis would be minimum as shown in Figure 3 and 4 which can be achieved. We investigate if we can identify C in Figures 3 and 4. One has the choice of identifying the basis by starting from an initial basis. There are 4 different ways of initializing the basis. (a) start at point A, (b) start at point B, (c) choose some $\mathcal{B}_{init}$ and (d) start with $\mathcal{B} = \text{null}$. We experiment with cases (c) and (d). Note that in both of these cases, we are traversing through only a portion of the curves shown in Figures 3 and 4 meaning starting at some point on the curve and reaching the knee point C.

Let $\mathcal{B}_{init}$ be an initial basis. Then a name $\{\mathcal{N}_p\}_{p=1}^{|\mathcal{N}|}$ in the database can, in *Case (a)*, be completely represented by using some of the elements in the basis, namely, $\mathcal{N}_p = b_i \oplus b_l \oplus ... \oplus b_m$, where $b_i, b_l, ..., b_m \in \mathcal{B}_{init}$ and in *Case (b)* be partially represented, namely, $\mathcal{N}_p = b_i \oplus n_b \oplus b_m$, where $b_i, b_m \in \mathcal{B}_{init}$, and $n_b \notin \mathcal{B}_{init}$. In *Case (b)*, for $\mathcal{N}_p$ to be representable using the basis, we need to necessarily add n_b to $\mathcal{B}_{init}$ and further make sure that n_b is orthogonal to all the elements in $\mathcal{B}_{init}$, namely $\{b_i\}_{i=1}^{|\mathcal{B}_{init}|}$. The addition of n_b introduces an extra element into $\mathcal{B}_{init}$, hence increasing size of the basis $|\mathcal{B}_{init}| + 1$. In reality we need to keep the size of basis as small as possible. The identification of an optimal basis set reduces to an optimization problem. Specifically, to optimize a function of $|\mathcal{B}|$ and $|\mathcal{J}|$. Namely,

$$\mathcal{C} = f(|\mathcal{B}|, |\mathcal{J}|) = |\mathcal{B}| \left\{1 + \frac{|\mathcal{J}|}{|\mathcal{N}|}\right\} \tag{3}$$

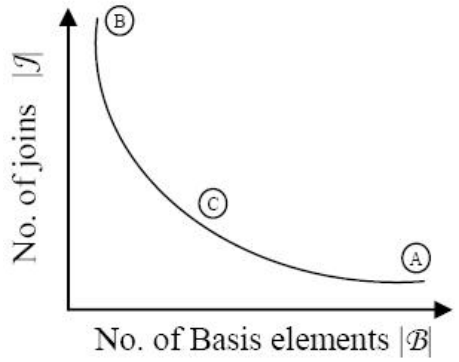

Fig. 3 Plot between number of elements in the basis, $|\mathcal{B}|$ and the total number of joins $|\mathcal{J}|$

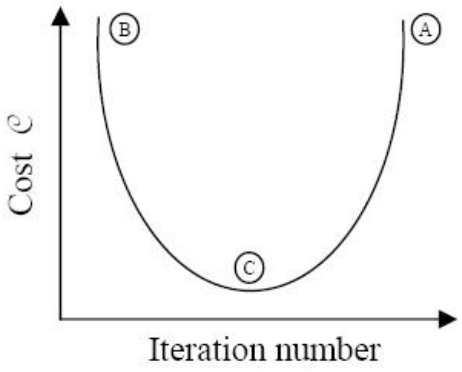

Fig. 4 Cost ($\mathcal{C}$) of constructing the pronunciation dictionary

where $\mathcal{C}$ is the cost of construction of the optimal basis $\mathcal{B}$. Figure 4 shows the variation of the cost function $\mathcal{C}$ for different combinations of $|\mathcal{B}|$[5] and $|\mathcal{J}|$. Now the optimization problem can be stated as

$$\min_{\mathcal{B}}\{\mathcal{C}\} \tag{4}$$

meaning, choose a basis $\mathcal{B}$ such that $\mathcal{C}$ is minimized. The object is to find the knee point C at which the cost ($\mathcal{C}$) of construction of the optimal basis ($\mathcal{B}$) corresponding to which the number of joins ($\mathcal{J}$) are "reasonable", would be minimum. The cost $\mathcal{C}$, would be maximum at the two extreme points A and B in Figures 3 and 4. At point A, $|\mathcal{B}| = 26$ and $|\mathcal{J}| = \sum_{i=0}^{|\mathcal{N}|}(l_i - 1)$ where l_i is the length of the name $\mathcal{N}_i$. In this case, as $|\mathcal{N}| \rightarrow \infty, |\mathcal{J}| \rightarrow \infty$ and hence $\mathcal{C} \rightarrow \infty$. At point B, $|\mathcal{B}| = |\mathcal{N}|$ and $|\mathcal{J}| = 0$. In this case, as $|\mathcal{N}| \rightarrow \infty$, $\mathcal{C} \rightarrow \infty$.

This formulation seeks the construction of an optimum basis that can span the entire database which can be achieved with optimal values for the two parameters $|\mathcal{B}|$ and $|\mathcal{J}|$ together. The expectation is that the optimum basis is created at some knee

[5] Here, $|\mathcal{B}|$ is not a basis in a strict sense as defined in Appendix A

point C on the curve shown in Figures 3 and 4, where the number of basis elements and the number of joins are optimal.

3 Algorithms for the construction of Basis

We propose two algorithms for the construction of the basis - (1) with a choice of initial basis (Algorithm 1), and (2) with out an initial basis (Algorithm 2).

Definitions

- Rank deficient basis: The basis set is called *rank deficient* if it is non-orthogonal meaning, some of the members in the set can be constructed using other entries in the set. A rank deficient basis does not satisfy the linear independence property (Appendix A).
- Span deficient basis: The basis set is called *span deficient* if it does not span the entire proper names database meaning, all the names in the database can not be constructed using this set. A span deficient basis does not satisfy the spanning property (Appendix A).

3.1 Algorithm 1

Step 1, `initialize` $\mathcal{B}_{init}$: The basis is initialized by sorting the $|\mathcal{N}|$ names in the database in the descending order of the number of occurrences in the database and then picking up all the names whose frequency of occurrence is greater than or equal to k% of the maximum frequency[6].
Step 2, `isOrtho(`$\mathcal{B}_{init}$`)`: $\mathcal{B}_{init}$ is checked for its orthogonality, namely, it is checked if any word in it can be completely constructed with a combination of other words in it. This task is accomplished by `isOrtho()`.
Step 3, `makeOrtho(`$\mathcal{B}_{init}$`)`: If $\mathcal{B}_{init}$ is found to be rank deficient (non-orthogonal), there is a need to make it orthogonal. If an element is found to be completely constructed with other elements in the set $\mathcal{B}_{init}$, that element is deleted from the set. This task is accomplished by a function named `makeOrtho()`. The process of orthogonalization of basis is described briefly in Appendix C.
Step 4: Start an iteration of constructing the basis.
Step 5, `initialize` $\mathcal{B}_m = \mathcal{B}_{init}$: We initialize a new set $\mathcal{B}_m$ with $\mathcal{B}_{init}$. $\mathcal{B}_m$ will be used to store the new words (that are not in $\mathcal{B}_{init}$) required to construct all the names in $\mathcal{N}$ if $\mathcal{B}_{init}$ is span deficient.
Step 6, `for each name` $\mathcal{N}_p \in \mathcal{N}$, we do the following.
Step 7, `forming` $\mathcal{B}_p$: Let $\mathcal{B}_p = \{b_{p1}, b_{p2}, ..., b_{|\mathcal{B}_p|}\}$ such that $\mathcal{B}_p \in \mathcal{B}$ and can completely or partially construct the name $\mathcal{N}_p$ in the database, namely, all the basis words in $\mathcal{B}_p$ are substrings[7] of the name $\mathcal{N}_p$. Note that $\mathcal{B}_p$ can be a null set meaning there are no elements in the basis which is a substring of $\mathcal{N}_p$. In such a case $\mathcal{N}_p$ should be added to the basis.

[6] 'Maximum frequency' refers to the frequency of the name which occurs the most number of times in the database.

[7] We consider a word as a substring of a name if it is a part of the name or sometimes the name itself

Algorithm 1 Pseudo-code

$\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_p, ..., \mathcal{N}_{|\mathcal{N}|}\}$
1. `initialize` $\mathcal{B}_{init} = \{b_1, b_2, \cdots, b_i, \cdots, b_{|\mathcal{B}_{init}|}\}$
2. `isOrtho`$(\mathcal{B}_{init})$
3. $\mathcal{B}_{init}$ = `makeOrtho`$(\mathcal{B}_{init})$
4. `do`
5. `initialize` $\mathcal{B}_m = \mathcal{B}_{init}$
6. `for each name` $\mathcal{N}_p \in \mathcal{N}$
7. {`form` $\mathcal{B}_p = \{b_{p1}, b_{p2}, ..., b_{pk}, ..., b_{p|\mathcal{B}_p|}\}$
`such that` $\mathcal{B}_p \subset \mathcal{B}_{init}$ and
b_{pk} `is a substring of` $\mathcal{N}_p$
8. `Identify all possible_sequences of`
$\mathcal{N}_p$ `using` $\mathcal{B}_p$ $\{\mathcal{N}_{p1}, \mathcal{N}_{p2}, ..., \mathcal{N}_{pl}, ..., \mathcal{N}_{pr}\}$
9. `collect all the new words`
10. `for each` $\mathcal{N}_{pl}$,
`obtain_cost` C_{pl} of $\mathcal{N}_{pl}$ (Equation 16)
`end for`
11. `choose` $\mathcal{N}_{pl}$ `with minimum` C_{pl}
`add new words in` $\mathcal{N}_{pl}$ to $\mathcal{B}_m$
`end for`
12. `isOrtho`$(\mathcal{B}_m)$
13. $\mathcal{B}_{init}$ = `makeOrtho`$(\mathcal{B}_m)$
14. `goto step` 5
15. `until`$(|\mathcal{B}_m| - |\mathcal{B}_{init}|) < \epsilon)$
16. $\mathcal{B}_{opt} = \mathcal{B}_{init}$

Step 8, construction of `possible sequences` ($\mathcal{N}_p$) `with` $\mathcal{B}_p$: Consider $\mathcal{B}_p$ is not empty; then the words in $\mathcal{B}_p$ may partially or completely construct $\mathcal{N}_p$ (see (1)). Let the name $\mathcal{N}_p$ be constructed with $\mathcal{B}_p$ in r different ways as shown in Figure 5.

As seen in Figure 5 there are r possible sequences constructed for the name $\mathcal{N}_p$. The choice of the l^{th} sequence is represented as $\mathcal{N}_{pl}$. For example, the sequence $\mathcal{N}_{p2}$ requires a new word n_{p21} which is not in $\mathcal{B}_p$, for successfully constructing $\mathcal{N}_p$, while $\mathcal{N}_{pl}$ requires two new words n_{p31} and n_{p32} to construct $\mathcal{N}_p$, while $\mathcal{N}_{p1}$ and $\mathcal{N}_{pr}$ completely construct the name without the aid of any new word being added to the existing basis. If more than one such representation of $\mathcal{N}_p$ is possible using different combination of words in $\mathcal{B}_p$, then a decision has to be taken as to which representation is to be retained. In such case, the selection depends on the cost of constructing the sequence.

Some sequences might partially construct the name. Or sometimes, none of the sequences might completely construct the name. In the later case, there is a need to include some new words into the basis, to enable the basis to construct the name (and the entry should also be orthogonal to the existing basis as mentioned earlier). So, a decision has to be taken about which new word(s) should be added to the basis and what is the cost of such addition.

Step 9, `collect all the new words`: The new words required by all the sequences formed for a name are collected and their frequency of occurrence is calculated. Note that even if more than one sequence formed for a name require a new word, the new word's frequency is counted only once. Thus the maximum value of frequency for a

$\mathcal{B} = b_1, b_2, ..., b_{|\mathcal{B}|}$
$\mathcal{B}_p = b_{p1}, b_{p2}, ..., b_{|\mathcal{B}_p|}$

$\mathcal{N}_{p1} = b_{p3} \oplus b_{p5} \oplus b_{p2}$
$\mathcal{N}_{p2} = n_{p21} \oplus b_{|\mathcal{B}_p|}$
.
.
.
$\mathcal{N}_{pl} = b_{p4} \oplus n_{p31} \oplus b_{p7} \oplus n_{p32}$
.
.
.
$\mathcal{N}_{pr} = b_{p5} \oplus b_{p6} \oplus b_{p4}$
where
$\mathcal{N}_{p1} = \mathcal{N}_{p2} = ... = \mathcal{N}_{pl} = ... = \mathcal{N}_{pr} = \mathcal{N}_p$

Fig. 5 Different ways of constructing $\mathcal{N}_p$ with $\mathcal{B}_p$

new word would be $|\mathcal{N}|$ meaning that this new word is required by all the names in the proper name database.

For every name in the database, we have several possible sequences and a list of words which are not in the basis. We need to choose one of the r sequence choices to represent the name $\mathcal{N}_p$. The choice is one that results in (a) minimal number of joins and (b) adds minimal number of entries to the existing basis set ($\mathcal{B}_{init}$). Observe that there is a need for optimality in choosing one of the r sequences. We construct a cost function to identify the optimal sequence choice.

Step 10, `obtaining cost for each word sequence` $\mathcal{N}_{pl}$: Let the l^{th} sequence ($\mathcal{N}_{pl}$) out of the r sequences that represent $\mathcal{N}_p$ has η_{pl} number of words, $\{s_1, s_2, ..., s_k, ..., s_n\}$ out of which η_{new} are new and η_{ex} belong to the existing basis, meaning $\eta_{pl} = \eta_{new} + \eta_{ex}$. If η_{joins} is the number of joins in the l^{th} sequence, then

$$\eta_{joins} = (\eta_{pl} - 1) \tag{5}$$

Let L be the length (number of letters) of the name $\mathcal{N}_p$. The cost function C_{pl} is formulated as

$$C_{pl} = f_{pl}(\mu, \nu, \eta_{pl}, \eta_{new}, \eta_{joins}, P_{av}, F_{av}, SA_{av}) \tag{6}$$

a function of the parameters of the word sequence $\mathcal{N}_{pl}$ namely, μ, ν, η_{joins}, η_{new}, η_{pl}, P_{av}, F_{av} and SA_{av}. And we choose the sequence $\mathcal{N}_{pl}$ such that

$$\mathcal{N}_{pl} = \underset{(l)}{\operatorname{argmin}}\{C_{pl}\} \qquad 1 \leq l \leq r \tag{7}$$

The cost function C_{pl} is best described by looking at each element involved in the construction of C_{pl}. We identify the relevance of the features and the redundancy in them in the following discussion. μ is the average length of the words in a sequence which is given by

$$\mu = \frac{1}{\eta_{pl}} \sum_{k=1}^{n_{pl}} l_k \tag{8}$$

where l_k is the length of the k^{th} word s_k in $\mathcal{N}_{pl}$. Maximization of μ reduces the number of joins in the sequence. Observe that the component $\sum_{k=1}^{\eta_{pl}} l_k = L$. Hence, (8) reduces to $\frac{\mu}{L} = \frac{1}{\eta_{pl}}$. So, maximization of μ means minimization of η_{pl} (number of words in the sequence) which in turn reduces the number of joins, η_{joins} (see (5)) in the sequence $\mathcal{N}_{pl}$. Hence, considering one of these three parameters is sufficient in formulating the cost function. ν is the variance of the lengths of words in the sequence $\mathcal{N}_{pl}$ and is given by

$$\nu = \frac{1}{\eta_{pl}} \sum_{k=1}^{\eta_{pl}} (l_k - \mu)^2 \tag{9}$$

P_{av} is the average percentage of acceptance of the words in the sequence $\mathcal{N}_{pl}$ and is given by

$$P_{av} = \frac{1}{\eta_{pl}} \sum_{k=1}^{\eta_{pl}} pd_k \tag{10}$$

where pd_k is the 'percentage demand' of the word from all the r sequences formed for the present name namely, pd_k is the percentage of r sequences formed for $\mathcal{N}_p$ which require s_k.

$$pd_k = \frac{Number\ of\ sequences\ demanding\ s_k}{r} \tag{11}$$

F_{av} which is defined only for the new words in the sequence, is the average frequency of occurrence of the new words in $\mathcal{N}_{pl}$ and is given by

$$F_{av} = \frac{1}{\eta_{new}} \sum_{k=1}^{\eta_{new}} f_k \tag{12}$$

where f_k is the frequency of occurrence of the new word s_k as a basis element, namely, f_k is the percentage of names in the database that are in requirement of s_k for their construction (even if one of the r sequences formed for $\mathcal{N}_p$ requires the word). So, f_k is given by

$$f_k = \frac{Number\ of\ names\ requiring\ s_k}{|\mathcal{N}|} \tag{13}$$

f_k of every new word is obtained in Step 9. SA_{av} is a binary valued attribute named by 'Syntax rule acceptance' and is defined for the new words in the sequence and checks if the word s_k to be introduced into the basis follows the syntactic rules given in Appendix B. sa_k is set to 1 if s_k follows the syntax rules and is set to 0 if it violates the syntax rules. η_{ak} is the number of words following the syntactic rules out of the η_{new} number of new words in the sequence $\mathcal{N}_{pl}$ (while the remaining are violating) and is given by

$$\eta_{ak} = \sum_{j=1}^{\eta_{new}} sa_k \tag{14}$$

SA_{av} is the percentage of new words following the syntactic rules given in Appendix B and is given by

$$SA_{av} = \frac{\eta_{ak}}{\eta_{new}} \tag{15}$$

Ideally, for any word in the sequence, the features l_k and pd_k should be maximum and for a new word to be included into the basis, its f_k should be maximum and sa_k should be 1. We saw earlier that considering one of the three features μ, n_{pl} and n_{joins} is sufficient. This implies that the 4 parameters μ, P_{av}, F_{av} and SA_{av} defined

for a word sequence should be maximized. In addition, the overall variance (ν) of the lengths of the words and the number of new words (η_{new}) in the sequence should be minimized for the sequence to be optimal. In other words, the proportionality of the cost of constructing a name (cost of selecting one of the r sequences formed for a name) with the features of the word sequence is given as follows

$$\begin{aligned} C_{pl} &\propto \eta_{pl} \\ &\propto \eta_{joins} \\ &\propto \eta_{new} \\ &\propto \nu \\ &\propto \frac{1}{\mu} \\ &\propto \frac{1}{P_{av}} \\ &\propto \frac{1}{F_{av}} \\ &\propto \frac{1}{SA_{av}} \end{aligned}$$

Considering the redundancy in features and their relation with the cost of construction of a name, we write the function f_{pl} as shown in (15).

$$f_{pl}(\mu, \nu, \eta_{new}, P_{av}, F_{av}, SA_{av}) = \frac{\lambda_\mu}{\mu} + \lambda_\nu \nu + \lambda_p P_{av} + \lambda_\eta \eta_{new}(\frac{1}{F_{av}} + \frac{1}{SA_{av}}) \quad (16)$$

where λ_μ, λ_ν, λ_p and λ_η are the weights assigned to μ, ν, P_{av} and the new words of the sequence respectively such that

$$\lambda_\mu + \lambda_\nu + \lambda_p + \lambda_\eta = 1 \quad (17)$$

We define the weight set as $\Lambda_1 = \{\lambda_\mu, \lambda_\nu, \lambda_p, \lambda_\eta\}$. Note that different choices of Λ_1 result in different basis sets. We choose that set which gives the minimal cost.

Step 11, `choose` $\mathcal{N}_{pl}$ `with minimum` C_{pl}: One sequence among the r sequences (formed for a name $\mathcal{N}_p$) which gives the minimum cost is selected (recall (7)) and the new words, if any, present in the sequence, are stored separately ($\mathcal{B}_m$ in the pseudo code). This process is repeated for all the proper names in the database. After all the names in the database are constructed with the existing basis, we are left with a set of new words to be introduced into the basis. We also have the frequency of occurrence of each of the new words and which database name is in requirement of a new word.

In summary, for a given name, the list of candidates from the existing basis that can construct the present name is collected and the sequences which partially or completely construct the name are formed. Based on the cost function formulated, one of the sequences that represent the name is selected and new entries, are made into the existing basis if required.

Step 12, `isOrtho(`$\mathcal{B}_m$`)`: After adding new elements to the existing basis (We add all the new words to $\mathcal{B}_m$ which is initialized to $\mathcal{B}_{init}$ - see Step 5), ($\mathcal{B}_m$) is checked for its orthogonality (rank deficiency).

Step 13, `makeOrtho(`$\mathcal{B}_m$`)`: If $\mathcal{B}_m$ is found to be rank deficient, it is made orthogonal using the function `makeOrtho()`.

By constructing the names in $\mathcal{N}$ with the existing basis (not in a strict sense), we check its *spanning property* and if it is found to be span deficient, by minimizing the cost function, we add to it, the required words to make it span the entire database. Then we check for its *orthogonality property* using `isOrtho()` and make it orthogonal using `makeOrtho()`.

This completes one iteration.

Step 14, `goto step 5`: Once, an orthogonal basis is formed, the database of names are again constructed with the updated basis. In this iteration, if some database names are not completely constructed with the pruned basis, some new entries are again made in to the basis based on the cost function formulated. The new basis is again checked for its orthogonality and pruned if necessary. The procedure of constructing the names of the database with the pruned basis is repeated again. New entries are appended to the basis if required.

Step 15, `until`$(|\mathcal{B}_m| - |\mathcal{B}_{init}|) < \epsilon$: The process of growing and pruning of the basis (checking for the spanning and orthogonality properties of the loosely defined basis set) is stopped when no significant growth and redundancy in the basis are observed in successive iterations. Note that ϵ is a small positive value.

Step 16, $\mathcal{B} = \mathcal{B}_{init}$: The optimum basis for the generation of pronunciation dictionary for the set of proper names, is the set of words obtained in the last iteration of pruning of the basis.

3.1.1 ***Conditions for convergence of the cost function***

We saw in Section 2 that the cost of constructing the basis is maximum at the points A and B in Figure 4 and an optimal basis is achieved at the knee point C. If $|\mathcal{J}_{init}|$ is the number of joins corresponding to the initial basis $|\mathcal{B}_{init}|$, then ($|\mathcal{B}_{init}|$, $|\mathcal{J}_{init}|$) is a point between the points A and C or B and C on the curve shown in Figure 3. The optimal basis $\mathcal{B}_{opt}$ is achieved at Step 14 in Algorithm 1 where the cost function converges to the knee point C. Note that in Algorithm 1, $\mathcal{C}$ is non increasing. Let the cost $\mathcal{C}$ at n^{th} iteration be $\mathcal{C}_n$ and at $(n+1)^{th}$ iteration be $\mathcal{C}_{n+1}$, then

$$\mathcal{C}_n = |\mathcal{B}_n| \left\{ 1 + \frac{|\mathcal{J}_n|}{|\mathcal{N}|} \right\}$$
$$\mathcal{C}_{n+1} = |\mathcal{B}_{n+1}| \left\{ 1 + \frac{|\mathcal{J}_{n+1}|}{|\mathcal{N}|} \right\}.$$

Convergence of the cost function is achieved when $\mathcal{C}_{n+1} \leq \mathcal{C}_n$ or $\mathcal{C}_n - \mathcal{C}_{n+1} \geq 0$ which reduces to

$$(|\mathcal{B}_n| - |\mathcal{B}_{n+1}|) + \frac{1}{|\mathcal{N}|} (|\mathcal{B}_n||\mathcal{J}_n| - |\mathcal{B}_{n+1}||\mathcal{J}_{n+1}|) \geq 0.$$

The cost function is convergent if and only if the conditions (18) and (19) are satisfied.

$$|\mathcal{B}_{n+1}| \leq |\mathcal{B}_n| \tag{18}$$
$$|\mathcal{B}_{n+1}||\mathcal{J}_{n+1}| \leq |\mathcal{B}_n||\mathcal{J}_n| \tag{19}$$

The convergence is validated through experiments in Section 4.

3.2 Algorithm 2

Step 1, `for each name` $\mathcal{N}_p \in \mathcal{N}$ we do the following.
Step 2, construction of all `possible sequences` **(**$\mathcal{N}_p$**)**: Here, we have no initial basis. We construct the name $\mathcal{N}_p$ in all possible ways in which it can be constructed with its substrings[8]. An example is possible sequences of the name 'gopal' are {`g opal, go pal, gop al, gopa l, g o pal, g op al, g opa l, go p al, go pa l, gop a l, g o p al, g o pa l, g op a l, go p a l, g o p a l`}. The sequences that have a single letter are excluded from this list.
Step 3, `obtaining cost for each word sequence` $\mathcal{N}_{pl}$: In this case, the cost function C_{pl} is based on the parameters μ, ν, P_{av} and SA_{av}[9] and the function f_{pl} is given by

$$f_{pl}(\mu, \nu, P_{av}, SA_{av}) = \frac{\lambda_\mu}{\mu} + \lambda_\nu \nu + \lambda_p P_{av} + \frac{\lambda_s}{SA_{av}} \tag{20}$$

where λ_μ, λ_ν, λ_p and λ_s are the weights assigned to μ, ν, P_{av} and the syntax of the words respectively such that

$$\lambda_\mu + \lambda_\nu + \lambda_p + \lambda_s = 1. \tag{21}$$

We define the weight set as $\Lambda_2 = \{\lambda_\mu, \lambda_\nu, \lambda_p, \lambda_s\}$.
Step 4, `choose` $\mathcal{N}_{pl}$ `with minimum` C_{pl}: One sequence among the r sequences (formed for a name $\mathcal{N}_p$) which gives the minimum cost is selected (recall (7)) and the words in the sequence are added to the basis. This process is repeated for all the proper names in the database.
Step 5, 6: The basis thus formed is then checked for its orthogonality and made orthogonal using the functions `isOrtho()` and `makeOrtho()`.

Algorithm 2 Pseudo-code

$\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_p, ..., \mathcal{N}_{|\mathcal{N}|}\}$
$\mathcal{B} = \{\}$
1. `for each name` $\mathcal{N}_p \in \mathcal{N}$
2. `form all possible_sequences of` $\mathcal{N}_p$
   $\{\mathcal{N}_{p1}, \mathcal{N}_{p2}, ..., \mathcal{N}_{pl}, ..., \mathcal{N}_{pr}\}$
3. `for each` $\mathcal{N}_{pl}$,
   `obtain_cost` C_{pl} of $\mathcal{N}_{pl}$
   `end for`
4. `choose` $\mathcal{N}_{pl}$ `with minimum` C_{pl}
   `add new words in` $\mathcal{N}_{pl}$ to $\mathcal{B}$
5. `isOrtho`($\mathcal{B}$)
6. $\mathcal{B}_{opt}$ = `makeOrtho`($\mathcal{B}$)
   `end for`

[8] substring is a part of the name
[9] The definitions of the parameters remain same as discussed in Algorithm 1.

4 Experimental Results and Discussion

4.1 Proper names database

For our experimentation, we used a database of proper names[10] which consisted of $1,63,600$ entries, majority of which are Indian names. Majority of the names were made up of two parts - a first name and a second name[11]. The first and surnames are considered as two different names and the duplicates are removed. So, to create a transcription dictionary one had to achieve transcription of these unique names. To test the performance of the proposed algorithm we further processed these unique names by removing names with two or less number of characters. This resulted in a set of $|\mathcal{N}|$ = 25884 unique names.

4.2 Basis construction

The following results are obtained by using Algorithm 1 for the construction of basis. The names are first sorted in the descending order of their frequency of occurrence in the entire database. All the names whose frequency is greater than or equal to 40% of the maximum frequency are taken as $\mathcal{B}_{init}$ which resulted in $|\mathcal{B}_{init}| = 225=$ (Step 1; Algorithm 1). Using `isOrtho()` and `makeOrtho()`, $\mathcal{B}_{init}$ is checked for its orthogonality (see Appendix A) i.e., the words in $\mathcal{B}_{init}$ that can be constructed from the other words in $\mathcal{B}_{init}$ are removed from it. The process of orthogonalization of basis is described in Appendix C. This resulted in $|\mathcal{B}_{init}| = 224$. This set is not strictly a basis because it doesn't satisfy the Spanning Property (Property 2, Appendix A). The unique names in the database are then constructed by joining the words in $\mathcal{B}_{init}$. For a given name in the database $\mathcal{N}_p$, the set of all names in the basis $\mathcal{B}_{init}$, which are the sub-words of $\mathcal{N}_p$, $\mathcal{B}_p = \{b_{p1}, b_{p2}, ..., b_{pk}, ..., b_{p|\mathcal{B}_p|}\}$ is first collected (Step 7, Algorithm 1). Many of the names in $\mathcal{N}$ had their substrings set, $\mathcal{B}_p$ empty. This is because the initial basis contains only 224 words out of which all are 'complete names'. The probability of their occurrence as a part of other names is hence very low. This resulted in many new words being appended to the initial basis $\mathcal{B}_{init}$ which resulted in $|\mathcal{B}_1| = 25476$. Using `isOrtho()` and `makeOrtho()` $|\mathcal{B}_1|$ is made orthogonal, and this reduced the size of $|\mathcal{B}_1|$ to 10435. A significant growth and reduction in the size of basis is observed in this iteration. The above process of growing and pruning (orthogonalization) of the basis is repeated for a few iterations till no significant growth or reduction in the size of basis is observed.

For the set of experiments conducted, we chose the weight set as $\Lambda_1 = \{0.4, 0.2, 0.1, 0.3\}$. These values satisfy (17) . We chose this weight set as follows. The basis is constructed for 286 combinations of weights of the features of the cost function (16), each weight taking a value from $[0, 0.1, 0.2, \cdots, 0.9, 1]$. The chosen weight set is the one which resulted in the minimal overall cost given by (3).

Recall the cost function defined in (3) which has to be minimized. Table 1 shows the basis size before and after orthogonalization (columns 2 and 3 respectively), the number of joins corresponding to the rank deficient basis (column 2) and the corresponding

[10] Company address book

[11] We did not distinguish between a first name and a surname for the set of experiments conducted. We believe that the proposed algorithm shows a better performance if we create separate basis for first names and surnames.

Table 1 $|\mathcal{B}|$ vs $|\mathcal{J}|$, variation of $\mathcal{C}$

I. No	$\lvert\mathcal{B}_m\rvert$	$\lvert\mathcal{B}\rvert$	$\lvert\mathcal{J}\rvert$	$\lvert\mathcal{B}_m\rvert\lvert\mathcal{J}\rvert$ $*10^{-8}$	$\lvert\mathcal{C}\rvert$
1	25476	10435	27614	7.03	23006
2	11131	6168	38570	4.29	16307.7
3	6549	5985	39629	2.59	15348.1
4	6064	5990	39654	2.40	15326.1
5	6053	5991	39654	2.40	15326.1

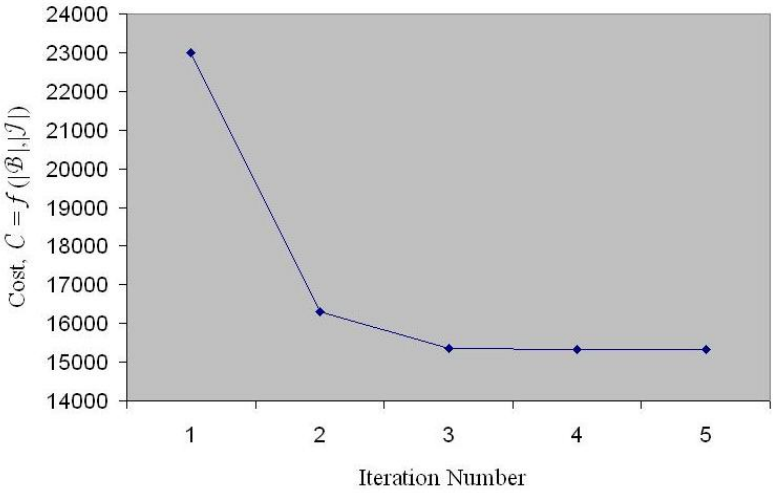

Fig. 6 Variation of $|\mathcal{C}|$ in (3) over 6 iterations

cost $\mathcal{C}$. Figure 6 gives the actual variation of cost function $|\mathcal{C}|$ over 5 iterations of constructing the basis until convergence. Figure 7 shows the plot between $|\mathcal{B}|$ and $|\mathcal{J}|$ for the experimentation performed which resembles Figure 3. We observe from Table 1 that the conditions (18) and (19) derived for the convergence of the cost function are met in the experimentation. We see from Column 3 of Table 1 that the size of basis is decreasing over iterations which satisfies condition (18). We see from Column 5 of Table 1 that the product of basis size and number of joins ($|\mathcal{B}_m||\mathcal{J}|$) is also reducing with iterations. The variation of the product of $|\mathcal{B}_m|$ and $|\mathcal{J}|$ until convergence is shown in Figure 8.

Compared to Algorithm 1, Algorithm 2 is simpler but computationally intensive. The results obtained by using Algorithm 2 are as follows. We chose $\Lambda_2 = \{0.4, 0.3, 0.3, 0\}$ based on experimentation. These values satisfy (17). The obtained values for basis size and number of joins are $|\mathcal{B}| = 7174$ and $|\mathcal{J}| = 38213$. The results presented for both the algorithms are obtained by not considering the feature SA_{avg} in the cost function.

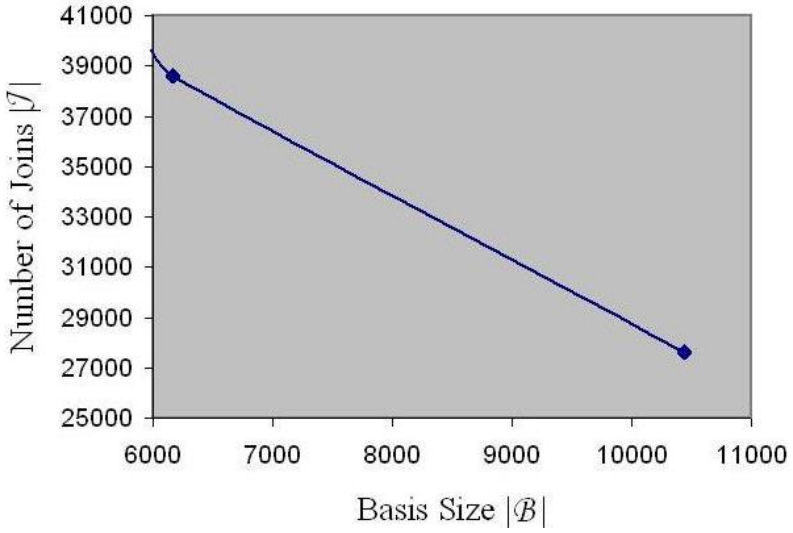

Fig. 7 Plot between basis size ($|\mathcal{B}|$) and total number of joins ($|\mathcal{J}|$)

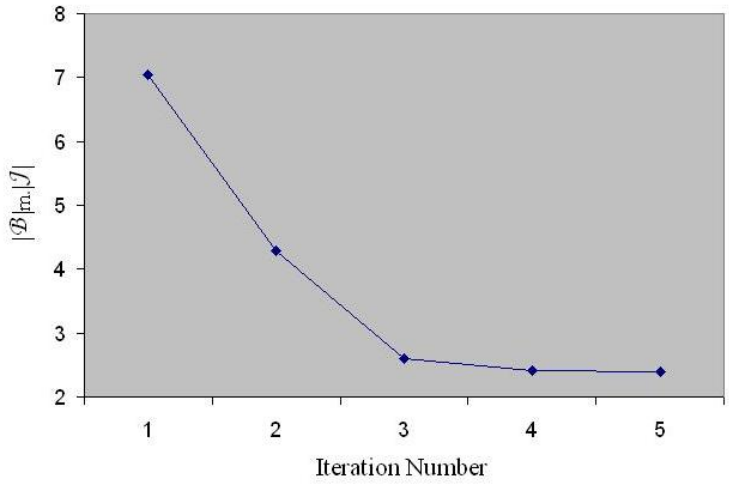

Fig. 8 Variation of $|\mathcal{B}_m||\mathcal{J}|$ over 5 iterations

4.3 Transcription of the optimal basis and the development of Pronunciation Lexicon

Now the obtained basis set has to be transcribed manually. We followed the following process which is specific to Indian languages. The basis words are written in Devanagari[12] (Hindi) script. Using a lookup table, which maps Devanagari graphemes to phonetic symbols, the Devanagari script of the basis words is converted into Festival TTS [14] accepted DARPA format and Microsoft supported SAPI format. For the sake

[12] a phonetic script

Table 2 A few Basis words and their transcriptions.

No.	Basis	DARPA	SAPI
1	kanth	k aa n th	k A n th
2	ma	m aa	m A
3	ra	r a	r a
4	je	jh ey	j E
5	shwar	s v ax r	S v a r
6	ram	r aa m	r A m

Table 3 Some proper names, their construction using the basis words, transcriptions in DARPA and SAPI formats and Festival TTS G2P transcriptions of the names.

Name	Word Seq	DARPA	SAPI	Festival (G2P rule base)
ramakanth	ra ma kanth	r a m aa k aa n th	r a m A k A n T h	r aa m ax k ae n th
rajeshwar	ra je shwar	r a jh ey s v ax r	r a j E S v a r	r ax jh sh w ao r
narendra	na ren dra	n ax r ey n dh r ax	n a r E n d r a	n r eh n d r ax
navyaram	navya ram	n ax v y aa r aa m	n a w y a r A m	n ae v y aa r ae m
kamlesh	kam le sh	k ax m l ey sh	k a m l E S	k ae m l ih sh

of consistency we had three people transcribe the basis separately and discuss among them to come up with a single set of transcription for all the basis words. One or more of these basis transcriptions are concatenated to generate the transcriptions of the proper names in the names database. Table 2 shows a few basis words and their corresponding transcriptions in DARPA and SAPI formats while Table 3 shows some proper names from the name database, their construction using the basis and the obtained phonetic transcriptions using the proposed algorithms.

4.4 Accuracy of the system

The next step is to ensure that the generated transcriptions obtained through manual transcription of the basis words are correct. The phonetic transcription of a proper name formed with two or more basis words will vary when a basis word has multiple possible pronunciations. This leads to a slight variation in the pronunciation of names. Note that the higher the occurrence of a basis word and the smaller its length, the greater is the possibility of variation in its pronunciation. The cost function formulated tries to maximize the length of the word which goes into the basis, thus minimizing the number of basis elements having multiple pronunciations. Nevertheless, the optimization process results in some basis words having multiple pronunciations which when used to generate transcriptions of names in the names database result in a slight variation in the pronunciation of names. They have to be taken care of manually when producing the final pronunciation lexicon of the names in the database. Our observation shows that the main source of multiple pronunciation arise in the presence of a vowel. While transcribing the basis words, sometimes, one may not be aware whether the vowel in the basis word is long or short. For example, in Table 3, the basis word `ra` in the first name `ramakanth` should have a transcription `r aa` and in the second name `rajeshwar`, it should be transcribed as `r a`. Using the same transcription of `ra` for both

the names leads to a slight variation in the pronunciation of one of them. However, for longer basis words the probability of multiple pronunciations is less.

After transcribing the basis, as mentioned above, 200 names from the proper names database are selected randomly and their phonemic transcription is constructed by concatenating one or more words using the transcribed optimal basis. The obtained transcriptions are verified manually and the number of correctly transcribed names are computed. This shows that 85% of the names are correctly transcribed. The inaccuracy in the transcriptions of the remaining 15% names is due to the multiple pronunciations of some basis words, majorly due to the following reasons: (a) long vowels in the basis words transcribed as short vowels (and vice versa) and, (b) multiple pronunciations of phoneme `/s/` and `/t/` which can have a phonetic form of [`/s/, /sh/`] and [`/t/,/T/`] respectively. The above is the case when different names with different pronunciations are spelled the same way. Also note that different people spell the same name in different ways inspite of having a unique pronunciation for the name. For example, an Indian name which has a phonetic spelling `ch au d a r i` is spelled in at least three different ways such as `chaudary, chowdhari, chaudhari` depending on a person's choice. Note that the proposed system generates slightly different transcriptions for these three instances of the same name.

4.5 Comparison with Festival TTS G2P rule base

Column 6 in Table 3 shows the phonetic transcriptions of the names obtained using the G2P rule base for the out of vocabulary (OOV) words used in the Festival TTS engine[13]. The results show that the transcriptions generated using the proposed method are found to be more accurate than the ones generated using the Festival TTS rule base. The proper names are also synthesized using these two kinds of transcriptions, (a) one obtained using the process discussed in this paper and (b) the one obtained using the G2P rule base used in Festival TTS, with the Festival TTS engine. The names synthesized using the transcriptions obtained by the proposed method are found to be perceptually better. The perception test was carried out by asking two persons who were not involved in the basis transcription process to listen to the synthesized proper names and rate the better of the two for each name (they had no idea which transcription was used in the synthesizing process.

Note: The algorithms proposed in this paper are generic and are suitable for proper names of any language. However, the system performs better if used for a database of proper names of same origin or geographical area. For example, the performance of the system is good when used to transcribe a proper names database containing only Indian names or only Chinese names, but degrades when used for a database which has a mix of both Indian and Chinese proper names. The results presented above are for a database containing a majority of Indian names but not all. Extending the same principle, if the system is used only for a database of person names or place names and not a mixture of both, the performance would be better.

[13] We used the Festival TTS G2P facility which was readily accessible

5 Conclusion

Research on automatic G2P transcription has reported promising results for phonetic transcription of regular text, where G2P transcriptions follow certain rules. Generating phonetic transcriptions of proper names, where the general purpose G2P converter can not be applied directly, involves human endeavor. In this paper, an optimization approach for the automatic generation of pronunciation lexicon for proper names has been proposed. We first construct a cost function and the transcription problem reduces to one of minimizing the constructed cost function. Two algorithms for the identification of basis have been proposed and the conditions for the convergence of the cost function have been derived. Experimental results on real database of proper names validate the convergence conditions derived and hence show that the developed optimization framework helps in reducing the mundane task of transcribing proper names. The formulated frame work is general and hence not restricted to Indian proper names, though the experimentation has been carried out on an Indian name database. In fact, the framework is suitable for any database of proper names irrespective of language. Through experimental results we have demonstrated the working and the validity of the proposed approach.

Acknowledgments

The authors express their gratitude to Amol, Meghna and Imran for their assistance in transcribing the basis and evaluating the generated phonetic transcriptions.

References

1. Thierry, Dutoit, High-quality Text-To-Speech synthesis: An Overview, Journal of Elec. and Electronics Engineering, Australia: Special Issue on Speech Recognition and Synthesis 17, 1 (1997) 25–37.
2. A. G. Ramakrishnan, M. Laxmi Narayana, Grapheme to phoneme conversion for Tamil speech synthesis, Proc. of Workshop in Image and Signal Processing (WISP-2007), IIT Guwahati (Dec 28-29 2007) 96–99.
3. Q. Yang, J.-P. Martens, N. Konings, H. van den Heuvel, Development of a phoneme-to-phoneme (P2P) converter to improve the grapheme-to-phoneme (G2P) conversion of names, in: Proceedings LREC, 2006, pp. 287–292.
4. B. Antonio, A. Jordi, D. A. Pablo, D. Erro, I. Esquerra, A. Moreno, J. Perez, T. Polyakova, The upc tts system description for the 2007 blizzard challenge, Proc. of Workshop in Image and Signal Processing (WISP-2007) The Blizzard Challenge 2007 – Bonn, Germany.
5. H. van den Heuvel, M. Jean-Pierre, K. Nanneke, (G2P) conversion of names. what can we do (better)?, INTERSPEECH 2007, Antwerp, Belgium (August 27-31) 1773–1776.
6. A. F. Llitjs, A. W. Black, Evaluation and collection of proper name pronunciations online, in: In Proceedings of LREC2002, Las Palmas, Canary Islands, 2002, p. 247254.
7. A. Font, Llitjos, A. Black, Knowledge of language origin improves pronunciation accuracy of proper names, Eurospeech, Aalborg, Denmark 3 (2001) 1919–1922.
8. A. F. Llitjos, A. W. Black, Knowledge of language origin improves pronunciation accuracy of proper names, in: In Eurospeech, 2001, pp. 1919–1922.
9. P. Onomastica, G. Joakim, Transcribing names with foreign origin in the onomastica project (1995).
10. T. Onomastica, Consortium, The onomastica interlanguage pronunciation lexicon (1995).
11. T. Toda, H. Kawai, M. Tsuzaki, Optimizing sub-cost functions for segment selection based on perceptual evaluations in concatenative speech synthesis, in: IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP apos;04)., Vol. 1, 17-21 May, 2004, pp. 657–660.

12. T. Toda, H. Kawai, M. Tsuzaki, K. Shikano, Perceptual evaluation of cost for segment selection in concatenative speech synthesis, in: Proceedings of IEEE Workshop on Speech Synthesis, 11-13 Sept 2002, pp. 183–186.
13. T. Toda, H. Kawai, M. Tsuzaki, K. Shikano, An evaluation of cost functions sensitively capturing local degradation of naturalness for segment selection in concatenative speech synthesis, Speech communication 48 (1) (2006) 45–56.
14. Centre for Speech Technology Research at The University of Edinburgh, Festival TTS, available at `http://www.cstr.ed.ac.uk/projects/festival/`.

A Basis

In linear algebra, a basis $\mathcal{B}$ of a vector space $\mathcal{V}$, by definition, is a set of linearly independent vectors that completely spans $\mathcal{V}$. $\mathcal{B} = \{b_1, ..., b_m\}$ is said to be a basis of vector space $\mathcal{V} = \{v_1, ..., v_n\}$ if $\mathcal{B}$ has the following properties:

- Linear independence property: If $a_1, ..., a_m$ are scalars and if $a_1b_1 + ... + a_mb_m = 0$, then necessarily $a_1 = ... = a_m = 0$. This implies that $b_1, ..., b_m$ are orthogonal or $b_1 \perp b_2 \perp ... \perp b_m$;
- Spanning property: For every v_k in $\mathcal{V}$ it is possible to choose scalars, $a_1, ..., a_n$ such that $v_k = a_1b_1 + ... + a_nb_n$.

B Syntactic Rules

It is advantageous to study/analyze the words syntactically before adding them in to the basis for if the resultant basis element is not following any syntax, its phonetic representation might not properly contribute to phonetically represent a longer name which is a super set of it. Syntactic knowledge is acquired by observing the sequences formed for a name. Some rules are illustrated below. V denotes a vowel and C denotes a consonant. If the word is of a particular format, the following decisions would be taken on its candidature for the basis. (Letters in bold represent the elements to be added to the basis).

- **CC reject**
  The pronunciation of a phone in a sequence of phones depends on the adjacent phones. Consonants depend on vowels for their pronunciation. So, the basis element cannot be a pure consonant sequence.
  Examples: `shashank sha+sha+nk`, `joseph - jose + ph`, `shantanu sha + nth + anu`, `sunny - su+nny`.
  All the words which are pure consonant strings are avoided. In other words, a basis element must have at least one vowel.
- **VC OK**
- **CV avoid**
- **VV reject**
  Introducing a split between two vowels is also not reasonable, because most of the times, the combination of two vowel letters in English forms a diphthongs. They may be two characters but their combination is a single sound. Example: shailendra - sh**a** + **i**lendra
- Introducing a split between sh, th, dh also should be avoided they are two characters but their combination is a single phone/sound
  Example: bharati bhara**t** + **h**i

C Orthogonalization of Basis

The following procedure is followed to make the basis orthogonal. Names in the basis are sorted in descending order of their lengths. For a word b_i in the basis, a set of all words b_{ik} which is a substring of b_i is collected, $\mathcal{B}_i = \{b_{i1}, b_{i2}, ..., b_{ik}, ..., b_{|\mathcal{B}_i|}\}$. If $\mathcal{B}_i$ is empty, b_i is retained in the basis. For the words whose $\mathcal{B}_i$ is not empty, elements of $\mathcal{B}_i$ are sorted in descending order of their lengths. One name b_{ik} from $\mathcal{B}_i$ is considered at a time and its position is fixed in the

word b_i. The remainder of b_i is filled with b_{ik} in the order they appear in $\mathcal{B}_i$. By the end of this process, b_i must have been formed completely or partially with the available elements in $\mathcal{B}_i$. With one b_{ik} at a time as the first element to occupy its place in b_i, and filling the remainder of the name with $\mathcal{B}_i$, we form $|\mathcal{B}_i|$ number of sequences for b_i. If any one of the $|\mathcal{B}_i|$ sequences completely represents b_i, then b_i is deleted from $\mathcal{B}_i$; else it is retained. The following example shows the sequences formed for the name, $b_i =$ `krishna`
$\mathcal{B}_i = \{$`krishn, krish, rish, kris, ris, ish, hna, na, kr, hn, is, ri, sh`$\}$

```
krishn     partially constructed
krish na     Fully constructed (1)
rish na     Partially constructed
kris hna     Fully constructed (2)
ris hna     Partially constructed
kr ish na     Fully constructed (3)
kris hna     Fully constructed (4)
krish na     Fully constructed (5)
kr ish na     Fully constructed (6)
kris hn     Partially constructed
kr is hna     Fully constructed (7)
ri hna     Partially constructed
kr sh na     Partially constructed
```

In the above example, the word `krishna` in the existing basis can be constructed in 7 different ways with the other existing basis elements. So, it is not necessary to have it in the basis and hence removed from the basis.