

# A Computational Phonetic Model for Indian Language Scripts

Anil Kumar Singh  
Language Technologies Research Centre  
IIT, Hyderabad, India  
anil@research.iiit.net

## Abstract

In spite of South Asia being one of the richest areas in terms of linguistic diversity, South Asian languages have a lot in common. For example, most of the major Indian languages use scripts which are derived from the ancient Brahmi script, have more or less the same arrangement of alphabet, are highly phonetic in nature and are very well organised. We have used this fact to build a computational phonetic model of Brahmi origin scripts. The phonetic model mainly consists of a model of phonology (including some orthographic features) based on a common alphabet of these scripts, numerical values assigned to these features, a stepped distance function (SDF), and an algorithm for aligning strings of feature vectors. The SDF is used to calculate the phonetic and orthographic similarity of two letters. The model can be used for applications like spell checking, predicting spelling/dialectal variation, text normalization, finding rhyming words, and identifying cognate words across languages. Some initial experiments have been done on this and the results seem encouraging.

## 1 Introduction

Most of the major Indian languages (Indo-Aryan and Dravidian) use scripts which have evolved from the ancient Brahmi script [24, 25]. There is a lot of similarity among these scripts (even though letter shapes differ). The letters have a close correspondence with the sounds. The arrangement of letters in the alphabet is similar and based upon phonetic features. If you list the letters on a paper, you can draw rectangles consisting of letters representing phonemes with specific phonetic features (voiced-unvoiced, etc). This well-organised phonetic nature makes it possible to build a computational phonetic model of these scripts. We present here such a model, which was needed urgently, especially because Indian languages have spelling variations even for common words. The scripts that we have covered are: Devanagari (Hindi, Marathi, Nepali, Sanskrit), Bengali (Bengali and Assamese), Gurmukhi (Punjabi), Gujarati, Oriya, Tamil, Telugu, Kannada, and Malayalam.

Brahmi origin scripts (also called Indic scripts) have been classified variously. Some of the terms used to classify these scripts are: syllabary, alphasyllabary and abugida. Out of these, abugida is perhaps the best term as it takes into account the property of these scripts which allows syllables to be formed systematically by combining consonants with vowel signs or *maatras*, rather than having unique symbols for syllables

which give no indication of the phonetic similarity among them. However, it should be noted that Brahmi scripts have properties that make their neat classification difficult [25]. They have an alphabet, they are syllabic, they have a way of forming syllables and they also have ‘featural’ properties in the sense that the position of a letter in the alphabet determines its features.

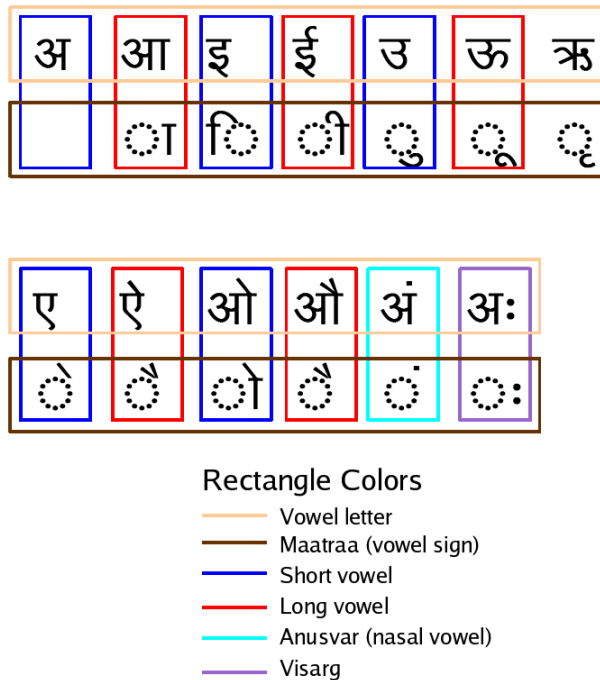


Figure 1: Phonetically arranged vowels in the alphabet

One terminological point can be made here. Even though syllable is a term used in describing the Brahmi scripts, there is another term (with very old origins) that better describes the units formed by combining the consonants and vowel signs. This term is *akshar* and is the one used in traditional Indian linguistics. There is some similarity between what the terms syllable and *akshars* mean, but there is a subtle difference, because of which the mapping between syllables and *akshars* is not exactly one-to-one. In our opinion, this difference can be stated as follows:

- Syllable is a psychologically real **phonological** unit
- *Akshar* is a psychologically real **orthographic** unit

This is the reason *akshar* is called ‘orthographic syllable’ by some scholars [25]. One simple example can illustrate this difference. The vowel *written* as ऐ or *E* may or may not be treated as a diphthong, depending on the context and also the language/dialect. In cases where it is treated as a diphthong, there is one *askhar* ऐ, but two syllables अ and इ. This difference will have some implications for building practical applications using the model proposed in this paper.

Still, the mapping between syllables and *akshars* is very nearly one-to-one, which is why abugida is a good enough term to describe Brahmi scripts. This fact also allows us

to build a compact computational phonetic model of scripts and use it easily for many practical (computational) applications.

## 2 Related Work

Emeneau [10], in his classic paper ‘India as a Linguistic Area’ showed that there are a lot of similarities among Indian languages, even though they belong to different families. This similarity was the major motivation for the work described in this paper.

There has been a lot of work on writing systems [6, 8, 27] from the linguistic point of view, but we will only mention some of the work related to computation here. Sproat [26] presented a computational theory of writing systems. He has studied the Brahmi scripts [24], and he also performed a formal computational analysis of Brahmi scripts [25].

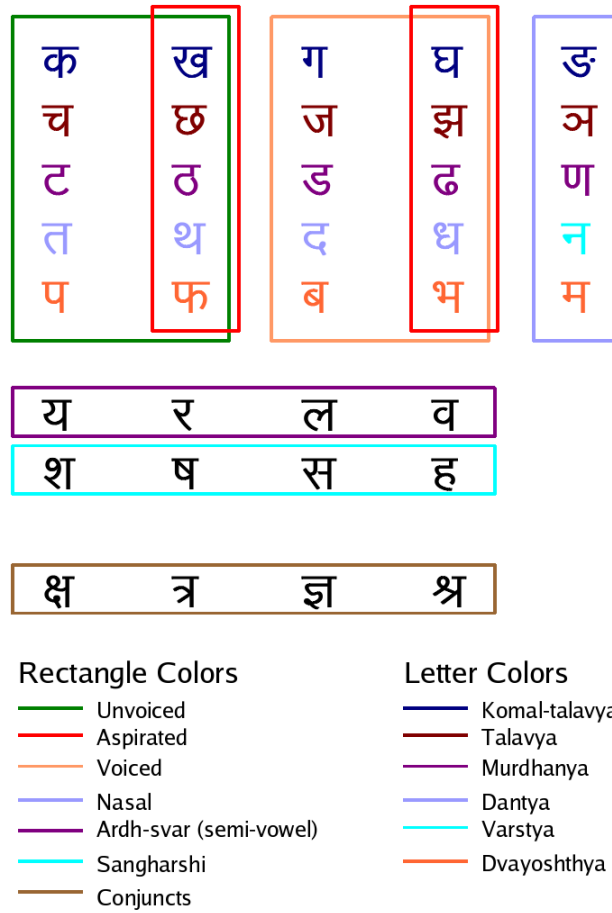


Figure 2: Phonetically arranged basic consonants in the alphabet

Another category of work is on alphabets and encodings for Indian languages. We will again focus on computation related work on alphabets and the encodings developed for them, especially for South Asian languages. Perhaps the most important work in this category is the development of a standard for Brahmi origin scripts [1, 4], called

क ख ग  
 ज  
 ङ ढ  
 न  
 य र ळ/ळ

Figure 3: Some additional phonetically arranged consonants in the alphabet

Indian Standard Code for Information Interchange (ISCII). This encoding, which has also been called a super-encoding [16] or meta-encoding, has not been successful in the sense that it has not been widely adopted for computing in South Asian languages, except in a handful of systems like the hardware solution called GIST [3]. In spite of its failure, it was an important work because it took into account the similarities among the alphabets of Brahmi origin scripts. This is why ISCII will be used as the basis for the ‘model of alphabet’ in this work, on which the phonetic model of scripts is based.

Another important work was the encoding standard UTF-8, which was itself based on Unicode [20, 13], UTF-8 being a transformation format for Unicode. The Indian languages section of UTF-8 is quite close to the ISCII standard, except that it has different slots for different scripts, unlike ISCII which is one unified encoding for all Brahmi origin scripts. Limitations have been pointed out in both ISCII and UTF-8 with respect to Indian languages. Still, in our opinion, both of them have very strong advantages over any other encoding for Indian languages. In practice, however, other encodings are preferred by users, simply because they have got used to these encodings. Most of these other encodings are either ad-hoc proprietary encodings or transliteration schemes. Encoding standards and support for them on computers are, in fact, the cause of a major bottleneck in computing for South Asian languages.

Coming to the work on phonetic modelling of graphemes, Reya et al. [23] argued that graphemes are perceptual reading units and can thus be considered the minimal ‘functional bridges’ in the mapping between orthography and phonology. Black et al. [2] discuss some issues in building general letter to sound rules within the context of speech processing. Galescu and Allen [11] present a statistical model for language independent bi-directional conversion between spelling and pronunciation, based on joint grapheme/phoneme units extracted from automatically aligned data. Daelemans and Bosch [7] describe another method for the same. Killer [14] has tried building a grapheme based speech recognition as a way to build large vocabulary speech recognition systems.

Sproat’s computational analysis of Brahmi scripts [25] was mainly oriented towards the shapes of the letters, i.e., graphemes. Kopytonenko et al. [15] also focussed on computational models that perform grapheme-to-phoneme conversion.

One recent work on the phonology-orthography interface was by Pandey [19] in which he argued that *akshar* is the the minimal articulatory unit of speech, but the focus of the work was to suggest proposals for reform in Hindi orthography.

There has also been a great deal of debate about issues like whether phonemes are

psychologically real or not, and if they are, whether they phonemes come from letters or letters come from phonemes. For example, Nootboom argues that phonemes come from letters [18], not vice-versa. The dominant view, in fact, seems to be tilting towards the view that ‘phonemes are useful fictions’ and their apparent psychological reality stems from the influence of orthography.

Without getting into the debate about primacy of phonemes or letters, we have tried to relate letters with phonetic and orthographic features in a way that allows some fuzziness. This is because phonemes are clearly fuzzy abstractions, irrespective of whether they are psychologically real or not. They are not clearly segmented discreet units in acoustic terms. We are basically trying to create a computational interface between letters and phonemes (or between their combinations) for Brahmi based scripts. For this, we are using orthographic and (mainly) phonetic features, a function for calculating the distance and an algorithm for aligning strings. This model tries to use linguistic knowledge about the writing systems used for South Asian languages.

### 3 The Need of a Phonetic Model of Scripts

There are several reasons why a phonetically (as well as orthographically) abstracted model of Brahmi origin scripts was needed. Some of these are:

- It is better to have one model and one text processing module for all the languages which use Brahmi scripts.
- There is much less standardization for these languages than for European languages, which means that the same word can have more than one valid spellings. It is not practically possible to list all the variants somewhere, e.g., in a dictionary. In fact, there may not be an agreement about valid and invalid spellings.
- Each of these languages has many dialects and there are dialectal variants of words. Even when writing the standard language, the writer may use a spelling (correct or incorrect) influenced by her first dialect or language. The dialectal variant may also be used deliberately by the writer for stylistic reasons or for representing the speech of some other person (say, a character in a novel or the person quoted in a news report).
- Due to reasons like low literacy rates and the use of English for many practical purposes (which means that many Indians, even those who are highly educated, may not have much practice in writing, let alone typing, in Indian languages), spelling errors are much more likely in the text written in Indian languages than they are for English.
- There are a large number of cognate words in Indian languages: words of Sanskrit origin (*tatsam* and *tadbhav* words), words borrowed from Persian, English, etc., words borrowed from one another. Some mechanism is needed to automatically identify these words.

## 4 Overview of the Model

The phonetic model tries to represent the sounds of Indian languages and their relations to the letters or *akshars*. It includes phonetic (articulatory) features, some orthographic features, numerical values of these features, and a distance function to calculate how phonetically similar are two letters, *akshars*, morphemes, words, or strings. Dynamic time warping (DTW) algorithm [17] is one of the algorithms that can be employed to align the strings for calculating the similarity (or distance) using the distance function. The phonetic model itself is based on a ‘model of alphabet’ which takes into account the similarities in the Brahmi origin scripts.

What this model tries to achieve can be summarized as:

- Enabling an abstract representation of text written in any Brahmi script
- Making it possible to calculate the orthographic and phonetic similarity between two letters, *akshars*, words, strings, etc.
- Ensuring that the representation is non-rigid and has an element of fuzziness so that it is possible to allow variations of the same word to be processed easily, even if not with 100% accuracy

Later in the paper we will discuss how these properties of the model allow us to build better applications for text processing in the context of Indian (or South Asian) languages.

These desired properties of the model are achieved through the following components:

- A phonological model of the alphabet in terms of phonetic or articulatory (and some orthographic) features
- Numerical values of features for calculating distance (table-2)
- ISCII codes mapped to sets of such features (table-3)
- UTF-8 codes mapped to ISCII codes for the case where the input text is in UTF-8
- A stepped distance function (figure-4)
- An algorithm (e.g., the DTW algorithm) to calculate similarity of two strings where each node of the string is a set of features with numerical values (a feature vector) (figure-5)

The DTW algorithm is not a core part of the model, as it can be replaced by another aligning algorithm for calculating the similarity between two strings where the strings are represented as sequences of nodes and each node can have a set of features with numerical values. In fact, there are many variations of the DTW algorithm itself. The main requirement of the aligning algorithm is that it should allow fuzziness while matching strings in terms of phonological and orthographic features. There are two components of our model which allow fuzziness: one is the SDF and the other is the aligning algorithm.

Note that we have mentioned the *phonological model* as a part of the *phonetic model of scripts*. The former refers to only a representation of the alphabet (and hence the letters) in terms of phonological features. The latter includes all the other components listed above.

अ	a	आ	A	इ	i	ई	I	उ	u	ऊ	U
ए	e	ऐ	E	ओ	o	औ	O	अं	aM	अः	ah
क	k	ख	kh	ग	g	घ	gh	ङ	ng		
च	c	छ	ch	ज	j	झ	jh	ञ	ny		
ट	T	ठ	Th	ड	D	ढ	Dh	ण	N		
त	t	थ	th	द	d	ध	dh	न	n		
प	p	फ	ph	ब	b	भ	bh	म	m		
य	y	र	r	ल	l	व	v				
श	sh	स	s	ष	S	ह	h				
क्ष	ksh	त्र	tr	ज्ञ	jny						

Table 1: The core part of the alphabet in a roman notation

## 5 Model of Alphabet

The concept of this ‘model’ has been shown in figures 1-3. In computational terms, using the ISCII encoding takes care of a part of this model. The basic idea is that the position of a letter in the alphabet determines its orthographic as well phonetic characteristics in most cases, even though there are exceptions. This alphabet is common, or more accurately, is a subsuming alphabet for all Brahmi origin scripts, irrespective of the shapes of the letters. Since the shapes of the letters are not being considered, we will not be using the term *grapheme*. Instead, we will be considering only letters. In this sense, a letter is an abstraction and a grapheme is an instance of a letter (or of some other unit of the script, e.g. a syllable or an *akshar*).

The letters (and words) have been shown in this paper in the Devanagari script or in a roman notation or both. Table-1 shows the core of the alphabet (vowel signs and some other letters are not shown) in a roman notation where capitalization of vowels represents longer length of vowels and *h* followed by a consonant indicates aspiration. IPA notation has not been used because the letters don’t actually have the same *absolute* phonetic values even for the same language (different dialects have different values), let alone across all the considered languages. What matters is that their *relative* values are similar across languages and the *absolute* phonetic values also have some similarity. The roman notation is based on the sounds of standard Hindi.

The phonetic nature of the alphabet can be seen in the following properties (figures 1, 2 and 3):

- Letters neatly arranged on phonetic basis
- Vowels and consonants separated
- Consonants themselves separated on the basis of phonetic features

- Rectangles can be drawn around consonants to represent articulatory features

The computational phonetic model tries to make explicit, in computational terms, the phonetic (as well as orthographic) characteristics of letters in this unified alphabet of Brahmi origin scripts. This is done by mapping the letters to a set of features.

## 6 (Mainly) Phonetic Features

The phonetic features that we have used are mainly the ones considered in modern phonetics, but we have also included some features specific to Indian language scripts. The terminology we have used is a bit tentative — a mixture of modern phonetics, traditional Indian phonetics, and computation oriented terms (svar1 and svar2). Some of the features are boolean (true or false). These are: *maatras* or vowel sign, *voiced*, *aspirated*, *hard*, *Devanagari*, *Bangla*, *Dravidian*, *diphthong*. Other features and their possible values are given in table-2.

Feature	Possible Values
Type	Unused (0), Vowel <b>modifier</b> (1), <b>Nukta</b> (2), <b>Halant</b> (3), <b>Vowel</b> (4), <b>Consonant</b> (5), <b>Number</b> (6), <b>Punctuation</b> (7)
Height (vowels)	<b>Front</b> (1), <b>Mid</b> (2), <b>Back</b> (3)
Length	<b>Short</b> (1), <b>Medium</b> (2), <b>Long</b> (3)
Svar1	<b>Low</b> (1), <b>Lower Middle</b> (2), <b>Upper Middle</b> (3), <b>Lower High</b> (4), <b>High</b> (5)
Svar2	<b>Samvrit</b> (1), <b>Ardh-Samvrit</b> (2), <b>Ardh-Vivrit</b> (3), <b>Vivrit</b> (4)
Sthaan (place)	<b>Dvayoshthya</b> (1), <b>Dantoshthya</b> (2), <b>Dantya</b> (3), <b>Varstya</b> (4), <b>Talavya</b> (5), <b>Murdhanya</b> (6), <b>Komal-Talavya</b> (7), <b>Jivhaa-Muliya</b> (8), <b>Svayantramukhi</b> (9)
Prayatna (manner)	<b>Sparsha</b> (1), <b>Nasikya</b> (2), <b>Parshvika</b> (3), <b>Prakampi</b> (4), <b>Sangharshi</b> (5), <b>Ardh-Svar</b> (6)

Table 2: Non-Boolean phonetic features: the feature codes are shown in bold face, and the numerical values of the features are given in parenthesis

We have tried to select features (and their values) for the computational phonetic model of scripts to include everything that:

- Makes one letter different from another
- Makes one letter similar to another

The design of these features, though it is based on articulatory features which have a physical basis, can perhaps be improved to remove redundancy and to take care of additional phenomenon in various languages and dialects for which Brahmi scripts are used.

ISCII	Letter	Possible Values
164	अ	type="v"/voiced="t"/length="s"/svar2="i"/svar1="d"/height="m"
...	...	...
177	औ	type="v"/voiced="t"/length="l"/svar2="i"/diphthong="t"/height="b"
...	...	...
179	क	type="c"/sthaan="k"/prayatna="s"
180	ख	type="c"/aspirated="t"/sthaan="k"/prayatna="s"
181	ग	type="c"/voiced="t"/sthaan="k"/prayatna="s"
182	घ	type="c"/voiced="t"/aspirated="t"/sthaan="k"/prayatna="s"
...	...	...
194	त	type="c"/sthaan="d"/prayatna="s"
195	थ	type="c"/aspirated="t"/sthaan="d"/prayatna="s"
...	...	...
213	श	type="c"/voiced="t"/aspirated="t"/sthaan="t"/prayatna="g"
214	ष	type="c"/voiced="t"/aspirated="t"/prayatna="g"/sthaan="t"/hard="t"
215	स	type="c"/voiced="t"/prayatna="g"/sthaan="v"
216	ह	type="c"/voiced="t"/aspirated="t"
...	...	...
218	ऌ	type="v"/maatras="t"/voiced="t"/length="l"/svar2="v"/svar1="l"
219	ॡ	type="v"/maatras="t"/voiced="t"/length="s"/svar2="s"/svar1="g"
...	...	...
250	ॠ	type="n"

Table 3: Mapping from ISCII codes to orthographic and phonetic features

## 7 Phonetic Representation of Letters

Two major standard text encodings for Indian languages are ISCII and Unicode (or UTF8). Both of them cover the languages mentioned above. While the former covers all of them in one set of codes from 161 to 255, the latter has different slots for them. The mapping between the two for a particular script is mostly one-to-one, though not completely. This is why we have taken ISCII (which has been called a super-encoding [16] or meta-encoding for Indian languages) as the basis of our representation.

In our model, we first map the UTF8 codes to ISCII codes for every script. Then we assign a (mainly) phonetic representation of each ISCII code in terms of the features. For example, vowel *o* and consonant *n* will be represented as:

ओ 176 → [type=v, voiced=t, length=s, svar2=m, svar1=m, height=b]  
न 198 → [type=c, voiced=t, sthaan=v, prayatna=n]

Features of some more letters are given in the table-3.

## Decision Tree Like SDF

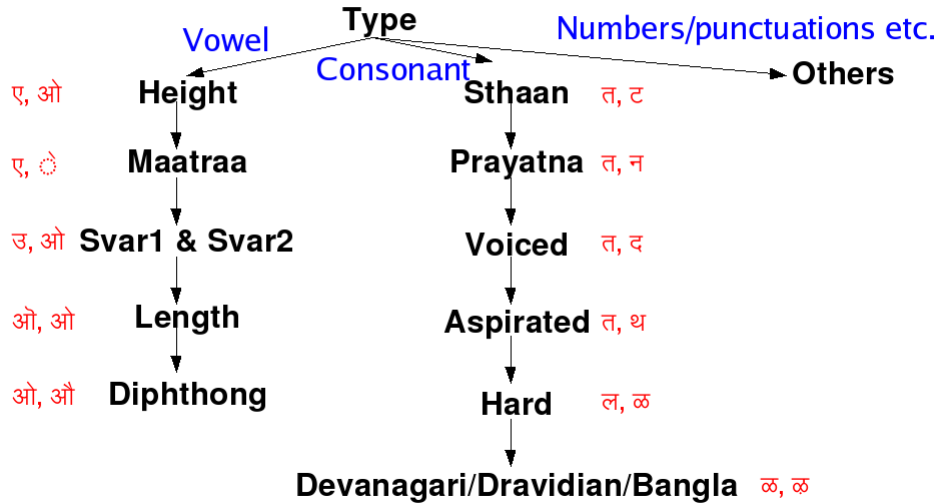


Figure 4: Decision tree like stepped distance function: various steps differentiate between different kinds of letters. The pairs of letters shown at each step are examples of such differentiation. At the end, a quantitative estimate of the orthographic and phonetic distance is obtained.

We also provide a numerical equivalent for every feature value such that it preserves the relationship between the values. For example, height (which can be front, mid or back) will have values 1, 2, or 3. Weights are also assigned to features. This allows us to take ISCII or UTF8 text in any of the languages, get its phonetic representation, and process it computationally. We can also calculate the phonetic distance between two letters or words in numerical terms.

## 8 Stepped Distance Function (SDF)

Ellison and Kirby [9] defined a new edit distance function which scales the Levenshtein distance by the average length of the words being compared. They used it for generating language taxonomies of Indo-European languages based on lexical surface similarity. Many other edit distance functions have been in use for applications like spell checking. In contrast to these methods, we use a more linguistically informed distance function.

To calculate the orthographic and phonetic similarity (or the distance) between two letters, we use a stepped distance function (SDF). Since phonetic features differentiate between two sounds (or letters representing them) in a cascaded or hierarchical way, the SDF calculates similarity at several levels. For example, the first level compares the type (vowel, consonant). There is a branching at the second level and, depending on whether the letters being checked are both vowels or consonants, further comparison is done based on the significant feature at that level: height in the case of vowels and *sthaan* (place) in the case of consonants. At the third level, values of *maatraa* and

*prayatna*, respectively, are compared. Thus, each step is based on the previous step. The weights given to feature values are in the non-decreasing order. The highest level (type) has the highest weight, whereas the lowest level (diphthong, for vowels) has the lowest weight. This process (somewhat simplified) is shown in figure-4.

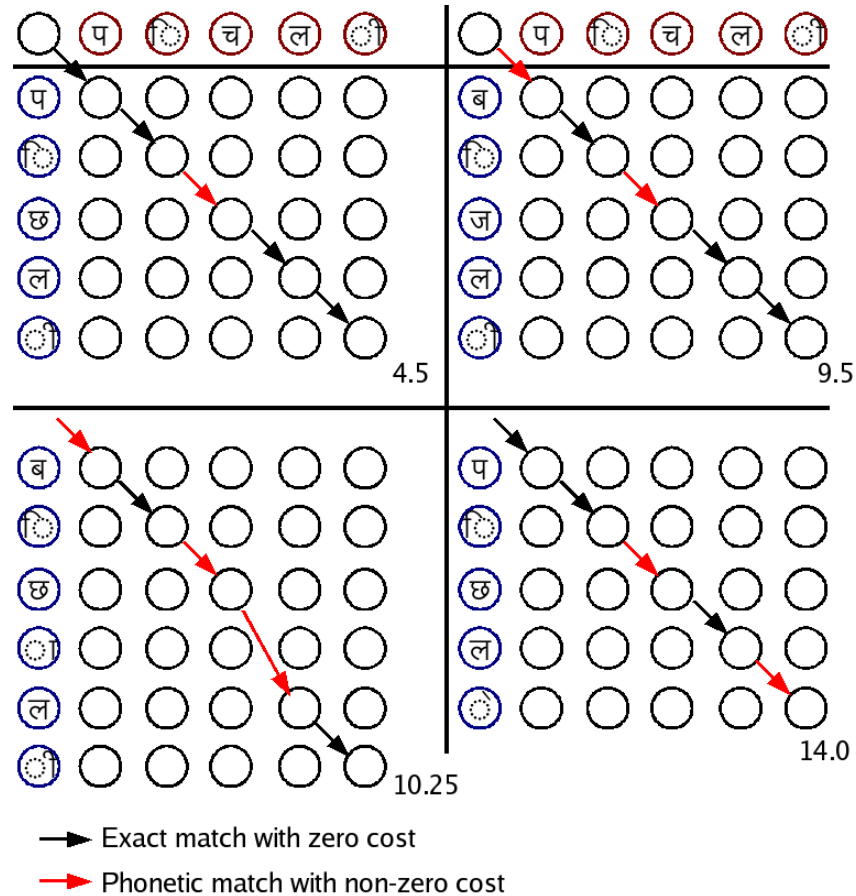


Figure 5: An example of aligning strings made up of phonetic and orthographic feature vectors, using the dynamic time warping (DTW) algorithm. The allowed arcs are: horizontal, diagonal, and double diagonal. The numbers shown are the final accumulated scores for the word pair.

## 9 Algorithm for Aligning Strings

One of the algorithms that can be used for alignment of strings is a simple version of the dynamic time warping algorithm used for isolated spoken word recognition (and also for aligning protein sequences, etc.). This algorithm can be used to align two strings made up of nodes. The nodes can be just letters, or they can be feature vectors (as in our case). There is a 'trellis', the two axes of which are the two sequences to be matched. The reference sequence is called the 'model' and the test sequence is called the 'data'. The data sequence is aligned with various model sequences and one or more models with the best scores are selected.

Nodes in the data are matched against the nodes in the model and accumulated costs (scores) are calculated. Since it is a dynamic programming algorithm, exhaustive search is not necessary. We can move from one node to another only by following the allowed arcs. These arcs represent operations like addition, deletion or substitution of nodes (letters in our case). Some cost is associated with various kinds of arcs. If the nodes match exactly, the cost is zero. In our implementation, we have only allowed horizontal, diagonal and double diagonal arcs. Since, for us, the nodes are vectors containing orthographic and phonetic features, the costs of arcs are actually calculated using the stepped distance function (SDF). The arc followed is the one associated with the lowest cost. At the end, we get the total accumulated cost (the distance or similarity score) for the data-model pair. This process is shown by an example in figure-5.

## 10 Generalising the Model for Other Scripts

One interesting question is whether this model can be applied to other scripts. This should not be much of a problem for scripts which have an alphabet and also have phonetic properties, e.g., Amharic. But for scripts like Roman/Latin as used for English (with notoriously unpredictable spellings: *ghoti* for fish, as Bernad Shaw noted), we will need more work. Still, it might be possible to modify the phonetic model for English. This could be done by one major change: instead of mapping a letter to a feature vector, we will have to map it to a two dimensional feature matrix. The first dimension will have the applicable feature values for that letter, while the second dimension will have the probabilities for the letter having that feature value. How will the probabilities be assigned? In our opinion, this can be done manually. Otherwise, if we have some ‘annotated’ corpus such as a pronunciation dictionary like those used for speech processing systems [5], we could even learn these probabilities automatically. The SDF will also have to be modified to calculate the distance in terms of probabilities of features.

## 11 Applications

We have tried this model for spell checking with good results. We are also working on using it for other applications like predicting spelling variations in Hindi (a link language in India) due to the influence of the native dialect or language of the writer, text normalization for other kinds of computational processing, etc. Some of these applications are briefly described in the following sections.

### 11.1 Spell Checker

Spell checking is the most obvious application of the phonetic model of scripts described in this paper. We simply need a reference word list (with correct spellings) and we calculate the similarity of some word in a document with the words in the reference list using the method described earlier. Then we can list some top few words, based on ranks or some threshold for similarity score. The second option seems better as different words in the document can have different number of closely matching reference words.

## Spell Checker Design

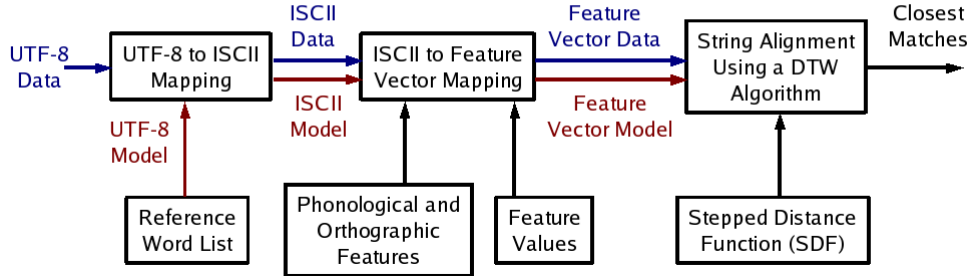


Figure 6: Design of a spell checker based on the phonetic model of scripts

The design of a spell checker based on the phonetic model is shown in figure-6. Sample output of the spell checker is shown in figure-7. Note that the usual string matching based algorithms for spell checking might not be able to give good scores to words which are similar phonetically but not orthographically. The spell checker based on the phonetic model can find such words. For example:

- पिछली (pichali) and बिजली (bijali)
- कलता (kalata) and करता (karata)
- तूम (tuma) and तुम (tuma)

The methods based on directly matching letters will not be able to see the similarity between प (p) and ब (b), between छ (ch) and ज (j), between ल (l) and र (r), or between तू (tu) and तु (tu). All of these pairs are phonetically close.

In the results shown in figure-7, the word रक्खा doesn't have an acceptable match (रखा) among those listed. This is simply because the acceptable entry was not present in the reference list of words.

The results shown in figure-7 were obtained by applying the following additional heuristics for matching:

- Either the beginning or the ending letter should match exactly.
- The difference in the lengths of the words being matched should be within a tolerance limit (of three letters in the setup used by us for testing).

These conditions can be tuned for getting the best results. The motivation for applying these conditions is the psycholinguistic fact that while reading we don't actually look at all the letters unless that becomes necessary. In most cases, the first letter, the last letter, and the length of the word are enough for us to predict what is the actual word. Since Indic scripts are not purely alphabetic, this fact may not be as much valid as it is for English (we have not experimentally verified this), but it can still be used to remove unlikely matches from the final list.

पिचली (picalI):		रक्खा (rakkhA):	
पिछली (pichalI)	4.5	रुक्का (rukkaA)	5.5
बिजली (bijalI)	9.5	सिक्का (sikkaA)	10.75
बिछाली (bichalI)	10.25	धक्का (dhakkaA)	18.5
पिछले (pichale)	14.0	रक्षा (rakshA)	20.5
संकठ (sankaTha):		फ्लेट (phleTa):	
संगठन (sangaThana)	4.75	प्लेट (pleTa)	4.5
संकट (sankaTa)	4.5	फ्लैट (phleTa)	10.5
संगठित (sangaThita)	4.75	फ्रेंच (phrenca)	13.0
संकेत (sanketa)	8.5	प्लान्ट (plAnTa)	16.0
रहैं (rahEM):		तूम (tUma):	
रहीं (rahIM)	8.5	धूम (dhUma)	5.5
हैं (hEM)	10.0	तुम (tuma)	7.5
रहें (raheM)	10.5	तुमने (tumane)	7.5
रहेंगे (raheMge)	10.5	तुमसे (tumase)	7.5
रहेंगी (raheMgI)	10.5	तुमको (tumako)	7.5
कलता (kalatA):		जूठा (juThA):	
कहलाता (kahalAtA)	2.0	झूठा (jhUThA)	3.5
करता (karatA)	5.25	जुटाने (juTane)	12.0
कराता (karAtA)	6.25	जुटा (juTA)	12.0
कर्ता (kartA)	6.25	टूटा (TUTA)	12.25

Figure 7: Sample output of the spell checker (phonetic and orthographic similarity scores)

## 11.2 Modelling and Tolerating Variation

Modelling, tolerating and predicting variation in spellings is an important issue for Indian languages as discussed in section-3. In this section we will describe an outline of a solution for this. The focus here is the framework for computational processing of phonological and spelling variation rather than on specific (statistical or rule based) techniques like hidden markov modelling (HMM). The idea is to have a framework under which we can experiment with various such techniques. The computational phonetic model is at the core of this framework.

This framework consists of the following components:

- A computational phonetic model of Indian language scripts as described in this paper.
- A ‘trie’ based compilation tool for lexicon to enable fast and easy search.
- A syllable or *akshar* forming component, as the Indian language scripts we are considering are syllabic in nature, in addition to having phonetic properties.
- A decoder which uses the lexicon ‘trie’ and the distance function to determine the standard or normalized form corresponding to the spelling variations. Note that the DTW algorithm described earlier also performs simple ‘decoding’ in this sense.
- A generator to predict the possible spelling variations given a standard or normalized form.
- A tool for studying spelling (and indirectly, phonological) variation given annotated or even unannotated corpora.

The decoder can use techniques like Gaussian mixture based HMM combined with dynamic time warping algorithm, as is done in speech recognition systems [28], but with a different implementation that avoids sequential search using the trie based lexicon compilation tool. The generator can be rule based or could use some statistical approach.

We conducted experiments on some words from two Hindi ‘regional’ novels by the famous author Renu [21, 22]. These novels have a lot of words with variant spellings which represent the locally used word forms for a particular ‘dialect’ of Hindi. One of the interesting and difficult cases was the name रामनारायण (rAmanArAyaNa). Two different forms of this name were used: रामलरैन (rAmalarEna) and रमलरैना (ramalarEnA). Both were aligned to the correct word form, i.e., रामनारायण at rank 2, the first being some other word. However, the correct form of another similar name सतलरैन (satalarEna), i.e., सत्यनारायण (satyanArAyaNa) was not listed among the best matching words. This shows that though the method we have used is fairly good at calculating phonetic and orthographic similarity, it still needs some tuning to take care of cases like the सतलरैन and सत्यनारायण.

Another interesting case was of जमाहिरलाल (jamAhiralAla) and जवाहरलाल (javAharalAla), which were aligned correctly with rank 1.

### 11.3 Text Normalization

This is similar to the previous application. The problem here is to replace all the variant forms with one standard form. Such ‘normalized’ text can then be processed much more easily by other NLP applications, because for many NLP applications keeping the variant forms as distinct word types may not be necessary. If we are able to take care of spelling variants (including dialectal variants), we can then find the best normalized form of a word.

This application will map more than one word forms to one standard word form. For example, रखा (rakhA), राखा (raakhA), रक्खा (rakkhA), रख्या (rakhyA), राख्या (raakhyA) should all map to रखा (rakhA). This will be very difficult to do using a rule based method, partly because listing the rules will be difficult. The phonetic model can allow us to specify the rules in an abstract form, as is done in texts on historical linguistics [12]. In fact, somewhat less accurate but quite flexible normalization can be achieved even by the method described for spell checking.

### 11.4 Finding Rhyming Words

Since we are calculating the phonetic as well as orthographic similarity, the words found to be similar will almost always be rhyming words. By tuning the implementation and allowing the user to do some customization on the conditions of matching, this method can give various kinds of rhyming words. In figure-7, the words रक्खा (rakkhA) and धक्का (dhakkA) are just rhyming words: they don’t have similar meanings and they have very little orthographic similarity.

### 11.5 Identifying Cognate Words Across Languages

As mentioned earlier, Indian languages have a lot of cognate words derived or borrowed from Sanskrit, English, Persian, and also from one another. Some examples of cognate words found by the current setup are:

- बालिस्टर (bAlisTara) and बैरिस्टर (bErisTara)
- तन्दरुसत (tandarusata) and तन्दुरुस्त (tandurusta)
- प्रेरना (preranA) and प्रेरणा (preraNA)

Some other cognate words which couldn’t be found are:

- इस्टाट (isTATa) and स्टार्ट (sTArTa)
- सिआणे (siaaNe) and सयाने (sayAne)

## 12 Future Work

To select the closest matches, since we are using a DTW algorithm, the search is sequential. This is not practical for applications like spell checking. A trie based lexicon compilation tool will overcome this problem.

Right now the SDF is used only to calculate similarity on letter-by-letter basis, we are working on generalizing it to calculate similarity on *akshar-by-akshar* basis. This will enable us to rule out many irrelevantly similar words. For example, words like रक्खा (rakkhA) and धक्का (dhakkA) are phonetically similar (rhyming), but this similarity may not be very relevant for applications like spell checking. Calculating *akshar-by-akshar* similarity may give such pairs low scores.

Another task will be to test the model for variation prediction and text normalization etc., using abstract rules of phonological transformation in terms of the features we have selected.

One obvious application of this model that we have not discussed in this paper will be to build a generalized text-to-speech system for Indian languages, but that will be a long term task.

### 13 Conclusions

Most of the major Indian languages use Brahmi origin scripts. These scripts have a lot of similarities. The computational (rather than formal) model of Brahmi origin scripts presented in this paper is a simple, easy to use model for practical NLP applications. Since it uses articulatory features, augmented with some orthographic features, to represent letters of a unified alphabet for Brahmi scripts, it is able to take care of spelling and dialectal/regional variations of word forms. This model consists of a mapping from UTF-8 codes to ISCII codes, a set of mainly phonetic features (including their codes and the numerical values assigned to them), a mapping from ISCII codes to feature vectors, a stepped distance function (SDF) for calculating the orthographic and phonetic similarity between two letters, and an algorithm (e.g., a dynamic time warping algorithm) for aligning strings made up of feature vectors as nodes. Some experiments have been performed using this model from the point of view of building applications like spell checking, text normalization, identifying cognate words across Indian languages, modelling and predicting spelling and regional/dialectal variation, and finding rhyming words. The core idea in all these applications is to calculate the orthographic and phonetic similarity between letters, words, strings, etc. The results are encouraging, but some practical issues remain, like search efficiency and tuning of features, the SDF and the aligning algorithm for handling difficult cases. The model presented in the paper could also be modified and used for other scripts.

Another possible application that we have not yet tried is for a generalised text-to-speech synthesis system for all Indian languages. This could be a topic for future research.

### References

- [1] BIS. Indian standard code for information interchange (iscii), 1991.
- [2] A. Black, K. Lenzo, and V. Pagel. Issues in building general letter to sound rules. In *ESCA Synthesis Workshop, Australia.*, pages 164–171, 1998.
- [3] C-DAC. About gist, 2006. <http://www.cdac.in/html/gist/about.asp>.

- [4] C-DAC. Standards for indian languages in it, 2006. <http://www.cdac.in/html/gist/standard.asp>.
- [5] CMU. The cmu pronouncing dictionary, 2007. <http://www.speech.cs.cmu.edu/cgi-bin/cmudict#about>.
- [6] Florian Coulmas. *Writing Systems: An Introduction to their Linguistic Analysis*. Cambridge University Press, 2003.
- [7] W. Daelemans and A. van den Bosch. A language-independent, data-oriented architecture for grapheme-to-phoneme conversion. In *Proceedings of ESCA-IEEE'94*, 1994.
- [8] P. Daniels and W. Bright. *The Worlds Writing Systems*. Oxford University Press, New York, 1996.
- [9] T. Mark Ellison and Simon Kirby. Measuring language divergence by intra-lexical comparison. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 273–280, Sydney, Australia, 2006. Association for Computational Linguistics.
- [10] M. B. Emeneau. India as a linguistic area. In *Linguistics* 32:3-16, 1956.
- [11] L. Galescu and J. Allen. Bi-directional conversion between graphemes and phonemes using a joint n-gram model. In *Proceedings of the 4th ISCA Tutorial and Research Workshop on Speech Synthesis*, 2001.
- [12] Hans Heinrich Hock. *Principles of Historical Linguistics*. Berlin: Mouton de Gruyter, 1986.
- [13] ISO. Utf-8, a transformation format of iso 10646, 2003. <http://www.ietf.org/rfc/rfc3629.txt>.
- [14] Mirjam Killer, Sebastian Stker, and Tanja Schultz. Grapheme based speech recognition, 2003.
- [15] Mykhaylo Kopytonenko, Keikki Lyytinen, and Tommi Krkkinen. Comparison of phonological representations for the grapheme-to-phoneme mapping. In *Constraints on Spelling Changes: Fifth International Workshop on Writing Systems*, Nijmegen, The Netherlands, 2006.
- [16] LDC. Found resources: Hindi, 2003. <http://lodl ldc.upenn.edu/found.cgi?lan=HINDI>.
- [17] C. S. Myers and L. R. Rabiner. A comparative study of several dynamic time-warping algorithms for connected word recognition. In *The Bell System Technical Journal*, 60(7), pages 1389–1409, 1981.
- [18] Sieb Nooteboom. Alphabets: From phonemes to letters or from letters to phonemes? In *Constraints on Spelling Changes: Fifth International Workshop on Writing Systems*, Nijmegen, The Netherlands, 2006.

- [19] Pramod Pandey. Phonology-orthography interface in devanagari script for hindi. In *Constraints on Spelling Changes: Fifth International Workshop on Writing Systems*, Nijmegen, The Netherlands, 2006.
- [20] Rob Pike and Ken Thompson. Hello world. pages 43–50, Winter 1993.
- [21] Phanishvar Nath Renu. *Mailaa Aanchal*. Rajkamal Paperbacks, 1954.
- [22] Phanishvar Nath Renu. *Parati Parikathaa*. Rajkamal Paperbacks, 1975.
- [23] Arnaud Rey, Johannes C. Ziegler, and Arthur M. Jacobse. Graphemes are perceptual reading units. In *Cognition* 74, 2000.
- [24] Richard Sproat. Brahmi scripts. In *Constraints on Spelling Changes: Fifth International Workshop on Writing Systems*, Nijmegen, The Netherlands, 2002.
- [25] Richard Sproat. A formal computational analysis of indic scripts. In *International Symposium on Indic Scripts: Past and Future*, Tokyo, Dec. 2003.
- [26] Richard Sproat. A computational theory of writing systems. In *Constraints on Spelling Changes: Fifth International Workshop on Writing Systems*, Nijmegen, The Netherlands, 2004.
- [27] Wikipedia. Writing system, 2006. [http://en.wikipedia.org/wiki/Writing\\_system](http://en.wikipedia.org/wiki/Writing_system).
- [28] Y. Zhao. A speaker-independent continuous speech recognition system using continuous mixture gaussian density hmm of phoneme-sized units. In *IEEE transactions on speech and audio processing*, 1(3), July 1993.