Sorting by Sound — Arbitrary Lexical Ordering for Transcribed Thai Text

Doug Cooper
<doug@chula.ac.th>
Center for Research in Computational Linguistics, Bangkok

When either Thai or transcribed (Romanized) Thai is sorted alphabetically, words that sound very much alike usually end up far apart. **maay** *and* **may** *are thrown to opposite ends of the letter* **m** *entries, even though mistaking one for the other causes problems for both foreign students who cannot speak clearly, and Thais who can't spell. This paper explains how and why the difficulty occurs, and shows why both Thai and transcription are inherently difficult to sort by sound. It introduces a method of preprocessing — deriving* ***phonemic signatures*** *— that lets us define improved lexical or dictionary orders, yet does not require anything but standard sorting code. The method can be applied to other languages — Lao, Khmer, and Burmese — that, like Thai, distinguish words on the basis of vowel length and/or tone.*

Introduction

Consider the dilemma of the Thai speller: these words are spelled thirteen different ways, but have essentially the same sound (*tʰan*), and vary only in tone and vowel length:

ทัน, ทันต์, ทันธ์, ธัญ, ถัน, ถั่น,
ทาน, ทานต์, ธาร, ถ่าน, ท่าน, ฐาน, ถาน.

There are a remarkable number of ways to spell words with this sound. Thai has six different *tʰ* letters, five ways to show *a* or *aa*, and six ways to write the final *n*. There are also four different tone marks, and a sign (over this letter ์) that means 'ignore me.' Finally, the tone mark does not actually give the tone — rather, it modifies an implicit tone that depends on the word's spelling.

Because a simple *one phoneme/one grapheme* (or *one sound/one letter*) relationship doesn't exist, words with identical or similar sounds can be widely scattered when lexically ordered. This complicates applications, ranging from 'sound-alike' spell checking to introductory language instruction, that depend on a Thai word's *sound*, not its spelling.

As a result, we find that sorting transcribed Thai is much more convenient that using native Thai orthography. We return to a relatively straightforward relationship between symbols and sounds; one that lets us group words with the same sounds regardless of their original Thai spelling.

Sorting transcriptions involves two basic issues: definition and implementation. First, we must define a *lexical* or *dictionary order*: if transcribed Thai adds the IPA symbols ə, ɛ, ɔ, ʉ to the English a, e, i, o, u, what should the combined set look like? Should may come before maay or vice versa? Should two-character symbols like k^h or p^h be removed from the midst of the k's and p's? What is the proper order of words that vary only by tone?

Second, we have to find an easy implementation — one that uses existing sort programs, even with a new character set or lexical order.

This paper looks at the issues involved in sorting by sound. Part I states the problem: it presents the terminology and issues of ordering, describes the difficulties of languages like Thai, and looks at questions that persist even with effective transcription systems.

Part II outlines the solution. It lists considerations for defining new lexical orders, then gives an algorithm for extracting *phonemic signatures* as part of a sorting strategy.

Finally, Part III deals with the implementation. It shows how to derive phonemic signatures, and uses simple UNIX tools to implement the algorithm for a test alphabet. The method is easily generalized to any consistent transcription scheme.

Part I: the Problem

Ordering, as opposed to sorting, relies on three sequences *collating, sorting*, and *lexical*.

— The *collating sequence* defines the order of the letters in a character set.

The ASCII set of 128 characters is the best known. Programs compare individual characters by their positions in the collating sequence; eg. *b* > *a* and *2* > *1*.

But because ASCII arbitrarily defines relations like *A* > *a* and # > *$*, the collating sequence does not reasonably answer questions like *what is the correct order of abAB?* There are six equally plausible possibilities: *abAB, ABab, aAbB, AaBb, AabB* and *aABb*.

— The *sorting sequence* overrides the collating sequence to put characters in a reasonable order regardless of relative positions. This is useful for sets that include *extended* or *upper ASCII* characters (eg. ISO Latin 1: A À Á Â Ã Ä Å Æ a à á â ã ...).

— *Lexical ordering* (or *dictionary ordering*) extends the sorting sequence by interpreting the meaning of characters.

In real dictionaries, this interpretation can be fairly sophisticated; eg. the number 9 may appear with the letter n, and punctuation is typically ignored.

Difficulty of Sorting by Sound in Thai

Sorting Thai by sound or tone turns out to be quite difficult. Reasons include:

— *Letters are not read in order as written.* In Thai, a vowel's sound frequently follows the next consonant. The word แก, which would be transliterated letter-for-letter as ɛɛk, is instead transcribed as kɛɛ.

— *There are more letters than sounds.* In Thai, the initial k^h (aspirated k) equivalent has three different letters devoted solely to it; t^h has six, s has four, etc. A final t can be written with eighteen different letters.

— *Tone production rules are not unique.* Thai derives tone from a combination of the orthographic and phonemic characteristics of opening and closing consonants, vowel length, as well as tone marks.

Difficulties of Ordering Transcriptions

The Haas method [1] is the best approach to Thai transcription. It relies on the *International Phonetic Alphabet* (IPA), and brings us close to the idyllic state — one grapheme per phoneme — that makes lexical ordering easy. I use it somewhat informally here; my apologies to the linguists in the audience. However, even Haas transcription has problems when it is used with standard sorting programs. We'll consider three of them.

The Ordering Problem The IPA is a set of supplemental symbols, rather than an ordinary alphabet. In effect, IPA is sorted by the coincidental overlap of your IPA font, and whatever standard set (eg. the Windows sort sequence, or the extended ASCII position numbers) the computer follows. For example, I use the freely available SIL Premier IPA fonts. Here's how the extended vowel symbols used to transcribe Thai fit in with the ordinary vowels. Neither sort has any visible logic, and neither matches the order I find easiest to remember personally.

a e i o u ɛ ə ɔ ʉ	*original list*
a ɛ e i ɔ o u ə ʉ	*Windows sort*
ɛ a e i o u ɔ ə ʉ	*UNIX (positional)*
a ɔ e ə ɛ i o u ʉ	*my preferred order*

Tone marks are scattered randomly through the character layout, and have the same problem. Their order makes no logical sense — lexical order must be organized externally.

The Intrusion Problem Transcribed Thai has 3 exceptions to a one-sound, one-letter rule:

- c^h, and the aspirated consonants k^h, p^h, and t^h are shown with two letters (possibly as ch, kh, ph, and th).
- long vowels are doubled, eg. e/ee or ə/əə.
- the glottal stop is shown with ?, either before or after the vowel.

The two-letter sounds cause a problem I call *intrusion* — a two-letter consonant can, and invariably does, appear in the middle of another consonant's dictionary section. For example, in Thai, initial p is distinct from initial (aspirated) p^h. However, an alphabetical sort yields:

pa	*through*	pe	. . . *then*
p^ha	*through*	p^hʉ	. . . *then*
pi	*through*	pʉ	. . . *and so on.*

The p entries are split in half by the intruding p^h. k, and t have equivalent problems, as do many of the glottal, short, and long vowels.

The Alternate Character Problem The undotted i character, ɪ, may be used to avoid conflicts with tone marks, eg.:

ìi, ǐi *vs.* ìi, ǐi

This wreaks havoc with lexical ordering. Because the undotted i character is in the position of " in standard character sets, ordinary sorting code handles ii and ɪi differently.

Tone marks also vary. The SIL IPA set has four versions of each tone mark, designed to fit appropriately around various characters. Once again, symbols with the same lexical position are found in different parts of the collating sequence.

ʉ̂ ʉ̂ ʉ̂ ʉ̂

Part II: the Solution

Thus, sorting Thai begins with transcription, then requires decisions about the order of:

- consonants, including IPA and two-letter consonants like ŋ and k^h.
- vowels, particularly IPA vowels ɛ ə ɔ ʉ.
- vowel length, eg. *glottal, short, long.*
- tones; for instance, Thai is conventionally ordered *mid, low, falling, high, rising.*

We must also come up with ways to:

- separate collating and sorting sequences,
- make two-character sequences sort as though they were single characters, and
- temporarily ignore tone marks or substitute characters that confuse sort programs.

Picking an Order for Sounds

Here's a restricted subset of the Thai alphabet that includes only the most regular of the duplicated consonant sounds:

กขงจชดฅทนบปพฟมยรลวสหอ◌ั ◌ุ ◌ิ ◌ี ◌ึ ◌ื

No Thai letter is used out of order. I can transcribe this 'alphabet' literally as:

kk^{h}ŋjc^{h}dtt^{h}nbpp^{h}fmyrlwsh?ɔaiʉueəɛo

Or, in an easier-to-remember arrangement, no English letter is out-of-order.

a͡ɔbcʰde͡ɘɛfhijkkʰlmnŋo͡ppʰrsttʰu͡ɯwy

Suppose we assume that short vowels precede longer ones. All that's left to define is tone order, which is traditionally:

` ^ ´ ˇ *mid, low, falling, high, rising*

Below, I've applied the rule *initial consonant / vowel length / tone* yields:

may	มัย	made of	1	*Thai*
may	ไมล์	mile	4	*dictionary*
mày	ใหม่	new	6	*order*
mây	ไม่	no, not	8	
mây	ไหม้	burn	9	
máy	ไม้	wood	5	
mǎy	ไหม	silk/question	7	
maay	มาย	measure, much	2	
mâay	ม่าย	widowed	3	
mâay	หม้าย	widow	11	
mǎay	หมาย	to intend	10	

Sorting on Phonemic Signatures

Next, we must transform the information we need — phonemes — into a form ordinary sort programs can use — single letters. Turning aʔ, a, aa, b, cʰ, d into A, B, C, D, E, F does the job, because the single upper-case letters are properly ordered in relation to each other.

Instead of editing the original words (that would destroy the useful information they contain) we extract information — the word's *phonemic signature*. If we prepend the signature to the word, and then sort, words will be in the order we seek:

Signature	*Original*	*Transformation*
At	aʔt	$a\text{ʔ} \Rightarrow A$
Bt	at	$a \Rightarrow B$
Ct	aat	$aa \Rightarrow C$
DBt	bat	$b \Rightarrow D,\ a \Rightarrow B$
EBt	cʰat	$c^h \Rightarrow E,\ a \Rightarrow B$
FBt	dat	$d \Rightarrow F,\ a \Rightarrow B$

When sorting is done, we throw the signatures away, and keep only the sorted originals.

More formally, we generate signatures and use them as sort *keys*. This approach is useful for sorting information, like fingerprint records, that does not easily lend itself to being ordered, and can also be applied to various programming problems (see [2]).

The example above has no tones, so one signature suffices. But multiple signatures let each signature act as the representative of a single sorting characteristic.

Suppose we have a number of characteristics — call them *S, T, U* — and each characteristic has alternatives, S_1, S_2, and so on.

S_1	T_1	U_1	$word_1$
S_1	T_2	U_1	$word_2$
S_1	T_2	U_2	$word_3$
S_1	T_2	U_3	$word_4$

All the words with characteristic S_1 group together. The S_1,T_n alternatives fall within this group, then the S_1,T_1U_n alternative (S_1,T_1U_1), and S_1,T_2U_n alternatives (there are three) follow.

In concrete terms we say that our signatures consist of *opening consonant and vowel, vowel length, closing consonant*, and *tone*, eg:

ma	1	y	1	máy	ไม้
ma	1	y	3	mày	ใหม
ma	2	y	3	mâay	ม่าย
ma	2	y	4	mâay	หม้าย

Each signature's influence depends on its position, left to right. Thai spellings, however different, are only considered if the signatures and the IPA transcriptions are identical.

Part III: Implementation

First, note the ASCII collating sequence (lowest to highest) is:

— *all white space, through single blank*

— ! " # $ % & ' ( ) * + , - . /

— 0 *through* 9, *then* : ; < = > ? @

— A *through* Z, *then* [\ ] ^ _ `

— a *through* z, *then* { | } ~

Deriving Signatures

A variety of techniques are employed to derive signatures from transcriptions:

— *Unifying* — giving different letters one value (so that ɪ and i might both be i).

— *Stripping* — removing characters (like tone marks) not relevant to a signature.

— *Compression* — turning a two-character sequence into a single character (eg. turning k^h and t^h into K and T).

— *Substitution* — giving individual characters more convenient names (eg. referring to ɛ and ʉ as E and U).

— *Remapping* — relocating non-contiguous characters (like stTuU...) to a sequence that sorts properly (eg. STUVW).

A few UNIX tools (also available as standalone utilities under DOS) suffice for all tasks.

An Example Alphabet

Consider a simple alphabet that displays all of the full alphabet's problems:

what we hear	d	ə	ɪ	i	k	k^h	n	ŋ	o	p	p^h	˜	ˆ
what we type	d	E	I	i	k	kh	n	m	o	p	ph	+	-

The transcription alphabet

Let's assume that we would like to define lexical ordering in the following way: characters are ordered as listed, single vowels follow doubled vowels, and tone marks appear in order following no tone marks.

The nonsense syllables below are properly ordered. The signatures are left in place; in a moment, we'll see how they were generated (they are the contents of file **s4**, below).

S3	*S2*	*S1*	*source*	*actual*
AB	AB	AB	dE	də
AB	AB	ABy	dE+	dă
AB	AB	ABz	dE–	dâ
AB	ABa	ABB	dEE	dəə
AB	ABa	AByB	dE+E	dăə
ABD	ABD	ABD	dEk	dək
ABD	ABD	ABzD	dE+k	dâk
ABD	ABaD	ABBD	dEEk	dəək
ABD	ABaD	AByBD	dE+Ek	dăək
DCI	DCaI	DCCI	kiip	kiip
DCI	DCaI	DCzCI	kI–ip	k̂iip
DHI	DHI	DHI	kop	kop
ECI	ECI	ECI	khip	kʰip

Signatures are generated right-to-left, starting with *source*. 'Extra' information is, in effect, ignored — it doesn't have to be stripped out.

— *S1* carries the tone-mark phoneme.

— *S2* strips tones, but carries the vowel length and final consonant phonemes.

— *S3* strips vowel length, but carries the initial consonant and vowel phonemes.

a marks a doubled vowel, and y or z mark tones. These particular values are chosen because they are greater, alphabetically, than the upper-case letters used for words. For example, BAAD precedes BAD, but BAD comes before BAaD.

If the alphabet includes a glottal stop, a slightly different strategy is used. Below, we rely on the fact that the digits precede the letters. All three vowel lengths are given a two-character signature; in *S2*, note that B9 is glottal, BB is normal, and Ba is long.

S3	*S2*	*S1*	*actual*
AB	AB9	ABy9	dă?
AB	ABB	ABy	dă
AB	ABa	AByB	dăə

Implementation Details

To generate the signatures, we begin with a wholesale transformation, then strip or modify individual characters as we build signatures.

```
source -> transliterate > s1
s1 -> strip-tones | tag-double-vowels > s2
s2 -> strip-vowel-tags > s3
assemble s3 s2 s1 source | sort > s4
s4 -> take-apart > out
```

All the intermediate files aren't necessary; I've left them to make the code easier to follow.

Transliteration The transliteration step requires several transformations. Some forethought is required in the transcription alphabet's design — if k and kh are each treated as a single letter, then h can't be a unique letter in the original alphabet *unless* k + h is illegal (as it is in Thai).

original	d	ə	ɪ	i	k	k^h	n	ŋ	o	p	p^h	ˇ	ˆ
source	d	E	I	i	k	kh	n	N	o	p	ph	+	–
compressed	d	E	I	i	k	K	n	N	o	p	P	+	–
unified	d	E	i	i	k	K	n	N	o	p	P	+	–
substituted	d	A	i	i	k	K	n	N	o	p	P	y	z
remapped	A	B	C	C	D	E	F	G	H	I	J	y	z

The transliteration step

The code below creates file **s1**:

```
sed 's/kh/K/g' < source |          compress kh
sed 's/ph/P/g' |                   compress ph
tr "IE+-" "iAyz" |                 unify and substitute
tr "dAikKnNopP" "ABCDEFGHIJ" > s1  remap
```

Strip-Tones, Tag-Double-Vowels Next we create file **s2** (my apologies for the **sed**):

```
tr -d yz < s1 |                    strip the tones
sed 's/\([BCH]\)\1/\1a/g' > s2     tag the doubled vowels
```

Strip-Vowel-Tags, Assemble, Take-Apart Next we'll strip the vowel tags and create file **s3**. Using auxiliary files in the next two steps makes the programming a bit clearer.

```
tr -d a < s2 > s3                  strip the vowel tags
```

Finally, we put the signatures together, sort, and cut the signatures out:

```
paste s3 s2 |                      paste the first two
paste - s1 |                       add the third
paste - source |                   add the original data
sort > s4                          do the work
cut -f4 < s4 > out                 strip off the first 3 fields
```

Further Work

The method presented here can be modified to accommodate different notions of appropriate phonemic sorting. We continue to work on additional questions, including:

- deciding what the most easily understood and used phonemic orders are,
- building electronic dictionaries / spelling assistants that are based on these orders,
- developing an easier user-interface for specifying how phonemic signatures should be extracted and evaluated,
- automatically generating the starting transcription.

References

[1] Haas, Mary. *The Thai System of Writing*. Spoken Language Services, Inc./American Council of Learned Societies, 1956.

[2] Bentley, Jon. *Programming Pearls*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.