Alignment of Multiple Languages for Historical Comparison

Michael A. Covington
Artificial Intelligence Center
The University of Georgia
Athens, GA 30602–7415 U.S.A.
mc@uga.edu

Abstract

An essential step in comparative reconstruction is to align corresponding phonological segments in the words being compared. To do this, one must search among huge numbers of potential alignments to find those that give a good phonetic fit. This is a hard computational problem, and it becomes exponentially more difficult when more than two strings are being aligned. In this paper I extend the guided-search alignment algorithm of Covington (*Computational Linguistics,* 1996) to handle more than two strings. The resulting algorithm has been implemented in Prolog and gives reasonable results when tested on data from several languages.

1 Background

The Comparative Method for reconstructing languages consists of at least the following steps:

1. Choose sets of words in the daughter languages that appear to be cognate;
2. Align the phonological segments that appear to correspond (e.g., skip the [k] when aligning German [knī] with English [niy] 'knee');[1]
3. Find regular correspondence sets (proto-allophones, Hoenigswald 1950);
4. Classify the proto-allophones into proto-phonemes with phonological rules (sound laws).

The results of each step can be used to refine guesses made at previous steps. For example, a regular correspondence, once discovered, can be used to refine one's choice of alignments and even putative cognates.

Parts of the Comparative Method have been computerized by Frantz (1970), Hewson (1974), Wimbish (1989), and Lowe and Mazaudon (1994), but none of them have tackled the alignment step. Covington (1996) presents a workable alignment algorithm for comparing two languages. In this paper I extend that algorithm to handle more than two languages at once.

2 Multiple-string alignment

The alignment step is hard to automate because there are too many possible alignments to choose from. For example, French *le* [lə] and Spanish *el* [el] can be lined up at least three ways:

```
e l     e l -     - e l
l ə     - l ə     l ə -
```

Of these, the second is etymologically correct, and the third would merit consideration if one did not know the etymology.

The number of alignments rises exponentially with the length of the strings and the number of strings being aligned. Two ten-letter strings have anywhere from 26,797 to 8,079,453 different alignments depending on exactly what alignments are considered distinct (Covington 1996, Covington and Canfield 1996). As for multiple strings, if two strings have A alignments then n strings have roughly A^{n-1} alignments, assuming the alignments are generated by aligning the first two strings, then aligning the third string against the second, and so forth. In fact, the search space isn't quite that large because some combinations are equivalent to others, but it is clearly too large to search exhaustively.

[1] These phonetic transcriptions may nor may not be phonemic. Because of the way the Comparative Method works, synchronic allophony is, in general, factored out along with diachronic allophony as the reconstruction proceeds.

Table 1: Evaluation metric used by Covington (1996).

Badness	Conditions
0	Exact match of consonants or glides
5	Exact match of vowels (nonzero so the aligner will prefer to match consonants, given a choice)
10	Match of 2 vowels that differ only in length, or [i] and [y], or [u] and [w]
30	Match of 2 dissimilar vowels
60	Match of 2 dissimilar consonants
100	Match of 2 unrelated segments
40	Skip preceded by another skip in the same string
50	Skip not preceded by another skip in the same string

Fortunately the comparative linguist is not looking for all possible alignments, only the ones that are likely to manifest regular sound correspondences – that is, those with a reasonable degree of phonetic similarity. Thus, phonetic similarity can be used to constrain the search.

3 Applying an evaluation metric

The phonetic similarity criterion used by Covington (1996) is shown in Table 1. It is obviously just a stand-in for a more sophisticated, perhaps feature-based, system of phonology. The algorithm computes a "badness" or "penalty" for each step (column) in the alignment, summing the values to judge the badness of the whole alignment, thus:

e	l
l	ə

$100 + 100 = 200$

e	l	-
-	l	ə

$50 + 0 + 50 = 100$

The alignment with the lowest total badness is the one with the greatest phonetic similarity. Note that two separate skips count exactly the same as one complete mismatch; thus the alignments

e	- e
l	l -

are equally valued. In fact, a "no-alternating-skips rule" prevents the second one from being generated; deciding whether [e] and [l] correspond is left for another, unstated, part of the comparison process. I will explain below why this is not satisfactory.

Naturally, the alignment with the best overall phonetic similarity is not always the etymologically correct one, although it is usually close; we are looking for a *good* phonetic fit, not necessarily the *best* one.

4 Generalizing to three or more languages

When a guided search is involved, aligning strings from three or more languages is not simply a matter of finding the best alignment of the first two, then adding a third, and then a fourth, and so on. Thus, an algorithm to align two strings cannot be used iteratively to align more than two.

The reason is that the best overall alignment of three or more strings is not necessarily the best alignment of any given pair in the set. Fox (1995:68) gives a striking example, originally from Haas (1969). The best alignment of the Choctaw and Cree words for 'squirrel' appears to be:

Choctaw	f a n i
Cree	- i l̥ u

Here the correspondence [a]:[i] is problematic. Add the Koasati word, though, and it becomes clear that the correct alignment is actually:

Choctaw	- f a n i
Koasati	i p - l̥ u
Cree	i - - l̥ u

Any algorithm that started by finding the best alignment of Choctaw against Cree would miss this solution.

A much better strategy is to evaluate each column of the alignment (I'll call it a "step") before generating the next column. That is, evaluate the first step,

-
i
i

and then the second step,

f
p
-

and so on. At each step, the total badness is computed by comparing each segment to all of the other segments. Thus the total badness of

a
b
c

is *badness*(a, b) + *badness*(b, c) + *badness*(a, c). That way, no string gets aligned against another without considering the rest of the strings in the set.

Another detail has to do with skips. Empirically, I found that the badness of

f
p
-

comes out too high if computed as *badness*(f, p) + *badness*(p, -) + *badness*(f, -); that is, the algorithm is too reluctant to take skips. The reason, intuitively, is that in this alignment step, there is really only *one* skip, not two separate skips (one skipping [f] and one skipping [p]). This becomes even more apparent when more than three strings are being aligned.

Accordingly, when computing badness I count each skip only once (assessing it 50 points), then ignore skips when comparing the segments against each other. I have not implemented the rule from Covington (1996) that gives a reduced penalty for adjacent skips in the same string to reflect the fact that affixes tend to be contiguous.

5 Searching the set of alignments

The standard way to find the best alignment of two strings is a matrix-based technique known as dynamic programming (Ukkonen 1985, Waterman 1995). However, dynamic programming cannot accommodate rules that look ahead along the string to recognize assimilation or metathesis, a possibility that needs to be left open when implementing comparative reconstruction. Additionally, generalization of dynamic programming to multiple strings does not entirely appear to be a solved problem (cf. Kececioglu 1993).

Accordingly, I follow Covington (1996) in recasting the problem as a tree search. Consider the problem of aligning [el] with [lə]. Covington (1996) treats this as a process that steps through both strings and, at each step, performs either a "match" (accepting a character from both strings), a "skip-1" (skipping a character in the first string), or a "skip-2" (skipping a character in the second string). That results in the search tree shown in Fig. 1 (ignoring Covington's "no-alternating-skips rule").

The search tree can be generalized to multiple strings by breaking up each step into a series of operations, one on each string, as shown in Fig. 2. Instead of three choices, match, skip-1, and skip-2, there are really 2×2: accept or skip on string 1 and then accept or skip on string 2. One of the four combinations is disallowed – you can't have a step in which no characters are accepted from any string.

Similarly, if there were three strings, there would be three two-way decisions, leading to eight ($= 2^3$) states, one of which would be disallowed. Using search trees of this type, the decisions necessary to align any number of strings can be strung together in a satisfactory way.

6 Alternating skips

Covington (1996) considers the alignments

e	- e
l	l -

equivalent and generates only the first of them, leaving it to some later step in the comparison process to decide whether [e] and [l] really correspond. The rule is:

> NO-ALTERNATING-SKIPS RULE: If there is a skip in one string, there cannot be a skip in the other string at the next step.

Although this tactic narrows the search space, I do not think this is linguistically satisfactory; after all, aligning [e] with [l] and skipping them in tandem are quite different linguistic claims. Consider for example the final segment of Spanish [dos] and Italian [due] 'two'; it is correct to skip the [s] and the [e] in tandem because they come from different Latin endings. It is not historically correct to pair [s] with [e] in a correspondence set.

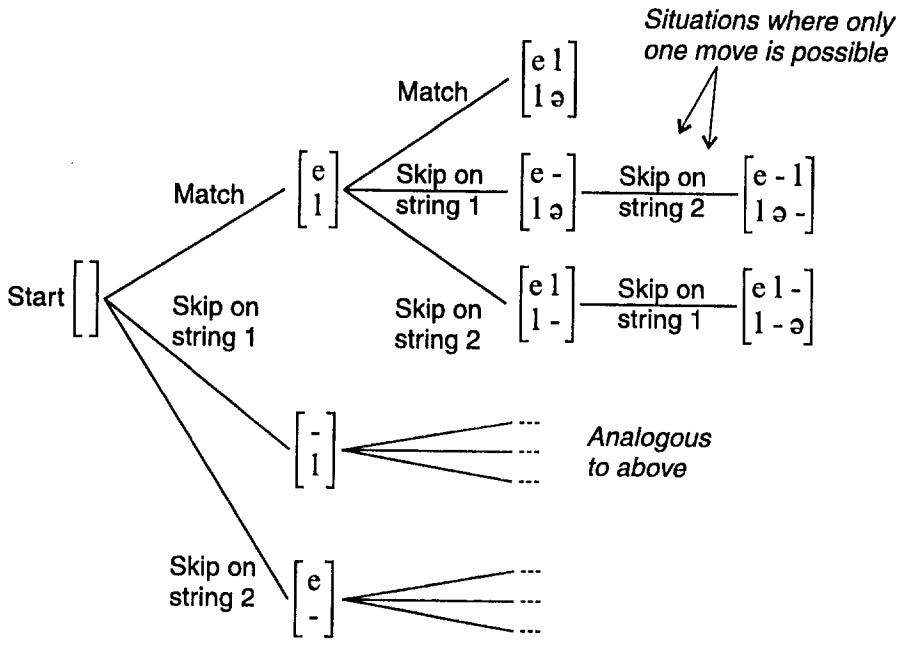

Figure 1: Part of a 3-way-branching search tree for generating potential alignments (Covington 1996, ignoring no-alternating-skips rule).

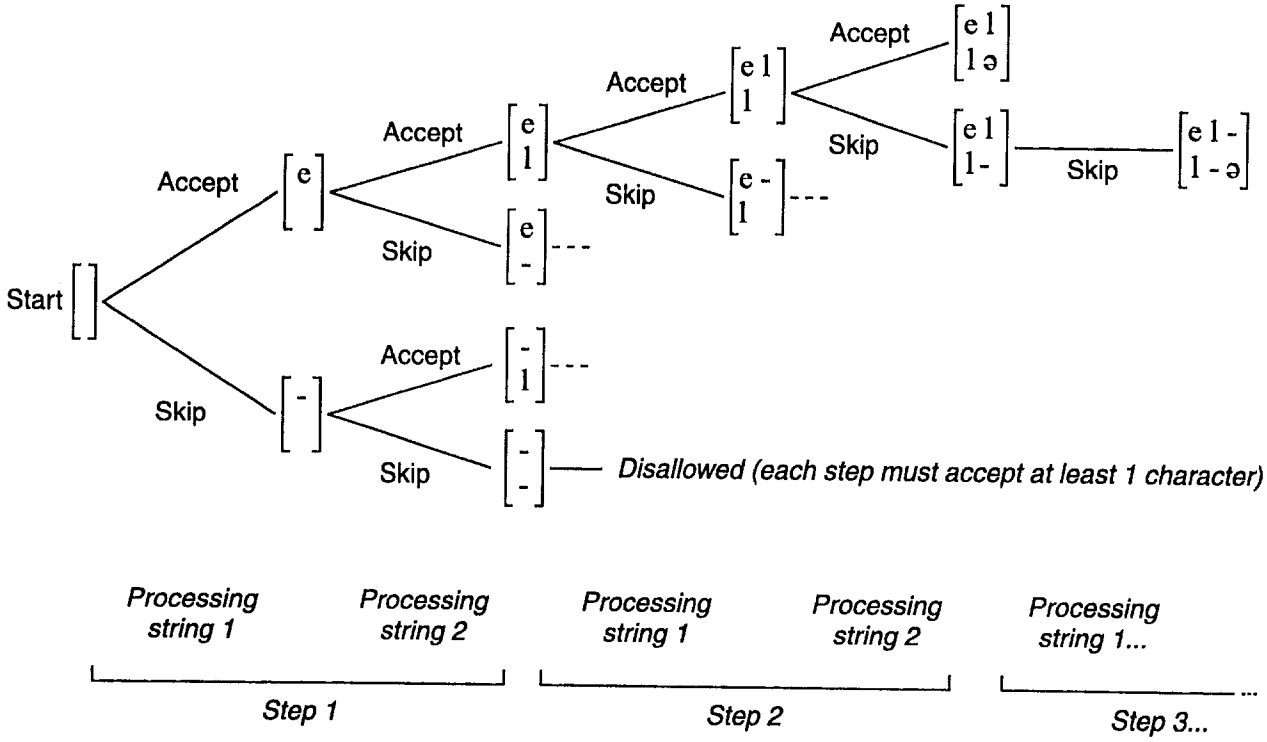

Figure 2: Search tree factored into 2-way branchings with a disallowed state at each step. This tree generalizes to handle more than 2 strings.

Also, the no-alternating-skips rule does not generalize easily to multiple strings. I therefore replace it with a different restriction:

> ORDERED-ALTERNATING-SKIPS RULE: A skip can be taken in strings i and j in successive steps only if $i \leq j$.

That lets us generate

```
- e          (String 1)
l -          (String 2)
```

but not

```
e -
- l
```

which is undeniably equivalent. It also ensures that there is only one way of skipping several consecutive segments; we get

```
- - - a b c
d e f - - -
```

but not

```
- a - b - c      a b c - - -
d - e - f -      - - - d e f
```

or numerous other equivalent combinations of skips.

7 Pruning the search

The goal of the algorithm is, of course, to generate not the whole search tree, but only the parts of it likely to contain the best alignments, thereby narrowing the intractably large search space into something manageable.

Following Covington (1996), I implemented a very simple pruning strategy. The program keeps track of the badness of the best complete alignment found so far. Every branch in the search tree is abandoned as soon as its total badness exceeds that value. Thus, bad alignments are abandoned when they have only partly been generated.

A second part of the strategy is that the computer always tries matches before it tries skips. As a result, if not much material needs to be skipped, a good alignment is found very quickly. For example, three four-character strings have 10,536 alignments (generated my way), but when comparing Spanish *tres*, French *trois*, and English *three*,[2] the algorithm finds its "best" alignment,

```
t r - e s
t r w a -
θ r - i y
```

after completing only ten other alignments, although it also pursues several hundred branches of the tree part of the way. (Here the match of [s] with [y] is problematic, but the computer can't know that; it also finds a number of alternative alignments.)

Table 2: Some alignments found by the prototype program.

```
Spanish/Italian/French 'three':
t r - e s
t r - e -
t r w a -

Spanish/Italian/French 'four':
k w a - t r o
k w a t t r o
k - a - t r -

Spanish/Italian/French 'five':
θ i ŋ k - o
c i ŋ k w e
s ɛ̃ - k - -

Koasati/Cree/Choctaw 'squirrel':
i p - l̥ u
i - - l̥ u
- f a n i
```

8 Results and evaluation

The algorithm has been prototyped in LPA Prolog, and Table 2 shows some of the alignments it found. None of these took more than five seconds on a 133-MHz Pentium, and the Prolog program was written for versatility, not speed.

As comparative linguists know, the alignment that gives the best phonetic fit (by any criterion) is not always the etymologically correct one. This is evident with my algorithm. For

[2] Admittedly an odd set to compare because of the different depth of branching, but they are cognates and each has four segments.

instance, comparing the Sanskrit, Greek, and Latin words for 'field,' the algorithm finds the correct alignment,

```
a g e r - -
a g - r o s
a ǰ - r a s        (badness = 365)
```

but then discards it in favor of a seemingly better alignment:

```
a g e r - -
a g - r o s
a - ǰ r a s        (badness = 345)
```

It doesn't know, of course, that [g]:[ǰ] is a phonetically probable correspondence.

Worse, occasionally the present algorithm doesn't consider the etymologically correct alignment at all because something that looks better has already been found. For example, taking the Avestan, Greek, and Latin words for '100', the algorithm settles on

```
- - s a t əm
h e k a t o n
k e n - t um        (badness 610)
```

without ever considering the etymologically correct alignment:

```
- - s a - t əm
h e k a - t o n
- - k e n t um      (badness 690)
```

The penalties for skips may still be too high here, but the real problem is, of course, that the algorithm is looking for the *one* best alignment, and that's not what comparative reconstruction needs. Instead, the computer should prune the search tree less eagerly, pursuing any alignment whose badness is, say, no more than 120% of the lowest found so far, and delivering all solutions that are reasonably close to the best one found during the entire procedure. Indeed, the availability of multiple potential alignments is the keystone of Kay's (1964) proposal to implement the Comparative Method, which could not be implemented at the time Kay proposed it because of the lack of an efficient search algorithm. The requisite modification is easily made and I plan to pursue it in subsequent work.

References

Covington, Michael A. (1996) An algorithm to align words for historical comparison. *Computational linguistics* 22:481–496.

Covington, Michael A., and Canfield, E. Rodney (1996) The number of distinct alignments of two strings. Unpublished manuscript, University of Georgia.

Fox, Anthony (1995) *Linguistic reconstruction: an introduction to theory and method.* Oxford: Oxford University Press.

Frantz, Donald G. (1970) A PL/1 program to assist the comparative linguist. *Communications of the ACM* 13:353–356.

Haas, Mary R. (1969) *The prehistory of languages.* The Hague: Mouton.

Hewson, John (1974) Comparative reconstruction on the computer. John M. Anderson and Charles Jones, eds., *Historical linguistics I: syntax, morphology, internal and comparative reconstruction,* 191–197. Amsterdam: North Holland.

Hoenigswald, Henry (1950) The principal step in comparative grammar. *Language* 26:357–364. Reprinted in Martin Joos, ed., *Readings in Linguistics I,* 4th ed., 298–302. Chicago: University of Chicago Press, 1966.

Kay, Martin (1964) *The logic of cognate recognition in historical linguistics.* (Memorandum RM-4224-PR.) Santa Monica: The RAND Corporation.

Kececioglu, John (1993) The maximum weight trace problem in multiple sequence alignment. *Combinatorial pattern matching: 4th annual symposium,* ed. A. Apostolico et al., 106–119. Berlin: Springer.

Lowe, John B., and Mazaudon, Martine (1994) The Reconstruction Engine: a computer implementation of the comparative method. *Computational Linguistics* 20:381–417.

Ukkonen, Esko (1985) Algorithms for approximate string matching. *Information and Control* 64:100–118.

Waterman, Michael S. (1995) *Introduction to computational biology: maps, sequences and genomes.* London: Chapman & Hall.

Wimbish, John S. (1989) *WORDSURV: a program for analyzing language survey word lists.* Dallas: Summer Institute of Linguistics.