

A heuristic for morpheme discovery based on string edit distance

John Goldsmith, Yu Hu, Irina Matveeva, and Colin Sprague

Keywords: morphology, unsupervised learning, string edit distance, Swahili

1 Introduction

This paper derives from work we have been doing on unsupervised learning of the morphology of languages with rich morphologies, that is, with a high average number of morphemes per word. Our focus in this paper is Swahili, a major Bantu language of East Africa, and our goal is the development of a system that can automatically produce a morphological analyzer of a text on the basis of a large corpus. While a certain amount of work in computational linguistics has already been done on Swahili, our specific goal is a system that can quickly and accurately perform a morphological analysis of any of the approximately 500 Bantu languages when presented with data from it, and little or no computational work currently exists for 99% of them. Our work reported here extends *Linguistica*, an open source system available at <http://linguistica.uchicago.edu>.

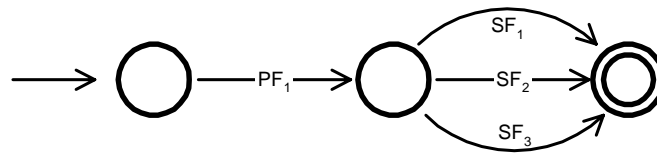
In this paper, we will present a new bootstrapping heuristic, one that is particularly useful in the analysis of languages with rich morphologies and that is based on the string edit distance dynamic programming algorithm (Wagner and Fischer 1974). We show that how it works and how it can be used to rank and quantify the robustness of morphological generalizations in a set of data. We test our bootstrapping algorithm against the current standard (derived from Harris (1955)) on a gold-standard of morphologically marked-up Swahili and evaluate the results.

Most systems designed to learn natural language morphology automatically can be viewed as being composed of an initial heuristic component and a subsequent explicit model. The initial or *bootstrapping* heuristic, as the name suggests, is designed to rapidly come up with a set of candidate strings of morphemes, while the model consists of an explicit formulation of either (1) what constitute an adequate morphology for a set of data, or (2) an objective function that must be optimized, given a corpus of data, in order to find the correct morphological analysis.

The best known and most widely used heuristic is due to Zellig Harris (1955) (see also Harris (1967) and Hafer and Weiss (1974) for an evaluation based on an English corpus), using a notion that Harris called successor frequency (henceforth, *SF*). Harris' notion can be succinctly described in contemporary terms: if we encode all of the data in the data structure

known as a trie, with each node in the trie dominating all strings which share a common string prefix,¹ then each branching node in the trie is associated with a morpheme break. For example, a typical corpus of English may contain the words *governed*, *governing*, *government*, *governor*, and *governs*. If this data is encoded in the usual way in a trie, then a single node will exist in the trie which represents the string prefix *govern* and which dominates five leaves corresponding to these five words. Harris's SF-based heuristic algorithm would propose a morpheme boundary after *govern* on this basis. In contemporary terms, we can interpret Harris's heuristic as providing *sets* of simple finite state automata (FSAs), as in (1), which generate a string prefix (PF₁) followed by a set of string suffixes (SF_i) based on the measurement of a successor frequency greater than 1 (or some threshold value) at the string position following PF₁.

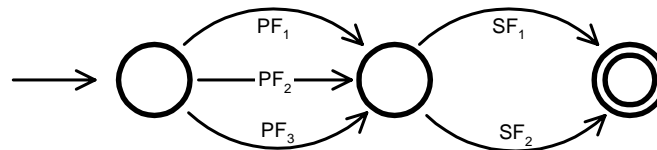
(1)



A variant on the SF-based heuristic, predecessor frequency (henceforth, PF), calls for encoding words in a trie from right to left. In such a PF-trie, each node dominates all strings that share a common string suffix. In general, we expect SF to work best in a suffixing language, and PF to work best in prefixing language; Swahili, like all the Bantu languages, is primarily a prefixing language, but it has a significant number of important suffixes in both the verbal and the nominal systems.

Goldsmith (2001) argues for using the discovery of signatures as the bootstrapping heuristic, where a *signature* is a maximal set of stems and suffixes with the property that all combinations of stems and suffixes are found in the corpus in question, and in Goldsmith (2004) this is explicitly tied to the Harris' SF heuristic. We interpret Goldsmith's signatures as extensions of FSAs as in (1) to FSAs as in (2); (2) characterizes Goldsmith's notion of signature in term of FSAs. In particular, a signature is a set of forms that can be characterized by an FSA of 3 states.

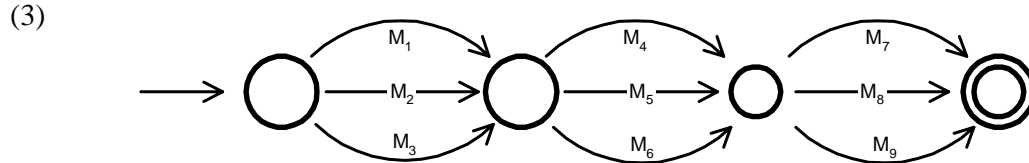
(2)



In this paper, we propose a simple alternative heuristic which utilizes the familiar dynamic programming algorithm for calculating string-edit distance, and finding the best alignment between two arbitrary strings (Wagner and Fischer 1974). The algorithm finds subsets of the

¹ We use the terms *string prefix* and *string suffix* in the computer science sense: a string S is a string prefix of a string X iff there exists a string T such that $X = S.T$, where "." is the string concatenation operator; under such conditions, T is likewise a *string suffix* of X. Otherwise, we use the terms *prefix* and *suffix* in the linguistic sense, and a string prefix (e.g., *jump*) may be a linguistic stem, as in *jump-ing*.

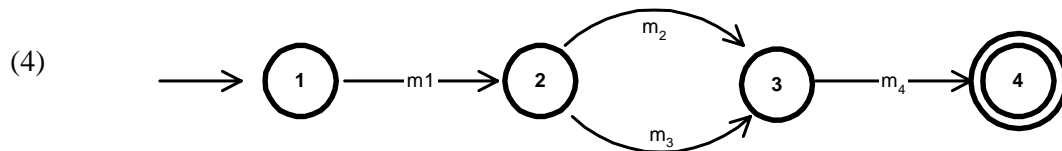
data that can be exactly-generated by sequential finite state automata of 4 states, as in (3), where the labels m_i should be understood as cover terms for morphemes in general. An automaton *exactly-generates* a set of strings S if it generates all strings in S and no other strings; a *sequential* FSA is one of the form sketched graphically in (1)-(3), where there is a unique successor to each state.



As we shall see, the informational significance and linguistic importance of such sequential FSAs can be easily calculated quantitatively, and thus we can rank the FSAs that emerge from the heuristic in terms of their importance for the data in the our corpus.

2 First stage: alignments.

If presented with the pair of strings *anapenda* and *anamupenda* from an unknown language, it is not difficult for a human being to come up with the hypothesis that *mu* is a morpheme inside a larger word that is composed of at least two morphemes, perhaps *ana-* and *-penda*. The same logic would lead to a similar conclusion when faced with the words *anamupenda* and *anawapenda*. Our method begins by utilizing the string edit distance (SED) algorithm to make this observation explicit, and by building small FSAs of the form in (4), where at most one of m_1 or m_4 may be null, and at most one of m_2 and m_3 may be null: we refer to these as *elementary alignments*. The strings m_2 and m_3 are called *counterparts*; the pairs of strings m_1 and m_4 are called the *context* (of the counterparts).



The first stage of our algorithm consists of looking at all pairs of words S, T in the corpus, and passing through the following steps:

We first pass over any words of length less than 5 letters, on the grounds that any similarities will too often be accidental.

Since the SED algorithm comparing two strings S and T is of $O(|S||T|)$ with a large coefficient, we look for ways to determine whether two strings are worth passing through the algorithm in the interest of saving considerable time. In principle we would like to set conditions on the *minimum* number of letters in the common spans in (4) (that is, $|m_1| + |m_4|$) and *maximum* number of letters in the $|m_2| + |m_3|$. In order to perform a rapid initial test that can precede (and in most cases, block) the SED algorithm, we test whether (a) the size of the intersection of the letters of the words, considered as a multiset, exceeds a threshold which we

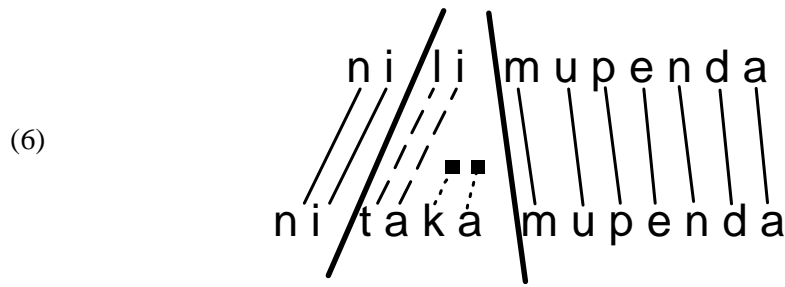
have set as 5, and we test (b) whether the size of the symmetric difference of the multisets of letters of the words exceeds a certain size, which we have set as 9.² Any pair (S, T) which fails (a) *or* passes (b) is eliminated, and we pass to the next pair of words.

Next, we compute the optimal alignment of S and T using the SED algorithm, where alignment between two identical letters (which we call *twins*) is assigned a cost of 0, alignment between two different letters (which we call *siblings*) is assigned a cost of 1.5, and a letter in one string not aligned with a segment on the other string (which we call an *orphan*) is assigned a cost of 1. An alignment as in (5) is thus assigned a cost of 5, based on a cost of 1.5 assigned to each broken line, and 1 to each dotted line that ends in a square box.



Since the standard SED algorithm does not allow for "line crossings" of alignments (i.e., for there to exist two non-negative integers i, j , and two positive integers m, n such that $S[i]$ is aligned with $T[j]$ and $S[i+m]$ is aligned with $T[j-n]$), it follows (given the costs specified above) that the alignments themselves (that is, the pairings of letters) are well-ordered. (The only non-obvious case to worry about, in the absence of alignment-crossings, is the case of letters that are not aligned at all; and it can easily be seen that if $S[i]$ and $T[j]$ are unaligned letters, then there must be two integers m and n such that $S[i+m]$ is aligned with $T[j-n]$; for if there were not, then $S[i]$ and $S[j]$ could be aligned without line-crossing at a lower total alignment cost.

To put the same point slightly differently, there is a natural map from every alignment to a unique sequence of pairs, where every pair is either of the form $(S[i], T[j])$ (representing either a twin or sibling case) or of the form $(S[i], 0)$ or $(0, T[j])$ (representing the orphan case). We then divide the alignment up into perfect and imperfect spans: perfect spans are composed of maximal sequences of twin pairs, while imperfect spans are composed of maximal sequences of sibling or orphan pairs. This is illustrated in (6).



² That is, the second value is equal to the sum of the lengths of the two strings minus the size of the intersections of the two strings viewed as unordered multisets of letters.

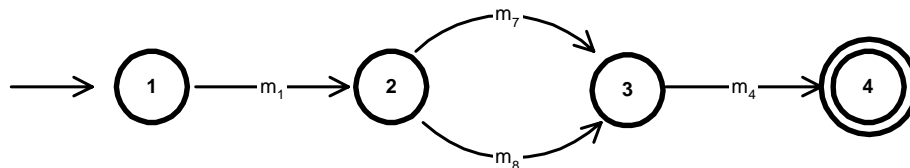
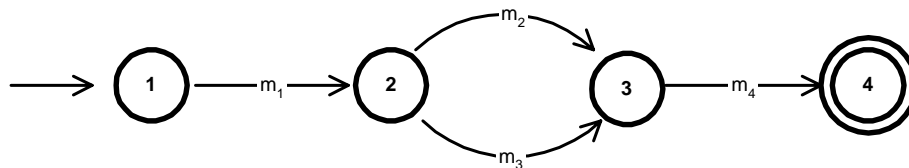
There is a natural equivalence between alignments and sequential FSAs as in (4), where perfect spans correspond to pairs of adjacent states with unique transitions and imperfect spans correspond to pairs of adjacent states with two transitions, and we will henceforth use the FSA notation to describe our algorithm.

3 Collapsing alignments

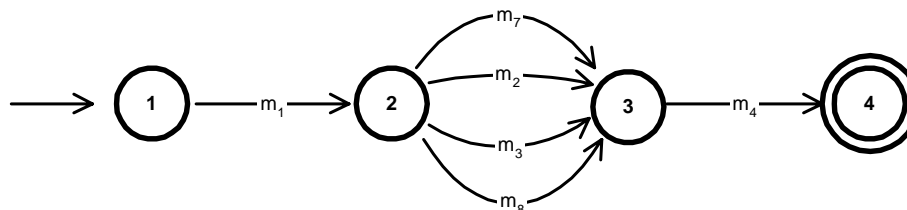
As we noted above (4), for any elementary alignment, a *context* is defined: the pair of strings (one of them possibly null) which surround the pair of *counterparts*. Our first goal is to collapse alignments that share their context. We do this in the following way.

Let us define the set of strings associated with the paths leaving a state S as the *production* of state S . A four-state sequential FSA, as in (4), has three states with non-null productions; if this particular FSA corresponds to an *elementary alignment*, then two of the state-productions contain exactly one string—and these state-productions define the *context*—and one of the state-productions contains exactly two strings (one possibly the null string)—this defines the *counterparts*. If we have two such four-state FSAs whose context are identical, then we collapse the two FSAs into a single *conflated* FSA in which the context states and their productions are identical, and the states that produced the *counterparts* are collapsed by creating a state that produces the union of the productions of the original states. This is illustrated in (7): the two FSAs in (7a) share a context, generated by their states 1 and 3, and they are collapsed to form the FSA in (7b), in which the context states remain unchanged, and the counterpart states, labeled ‘2’, are collapsed to form a new state ‘2’ whose production is the union of the productions of the original states.

(7) a.



b.

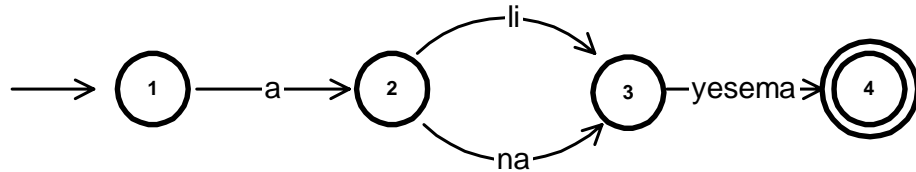


4 Collapsing the resulting sequential FSAs

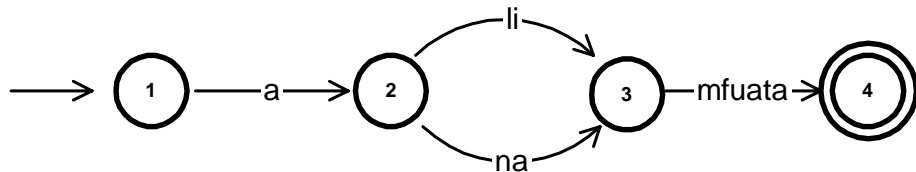
We now generalize the procedure described in the preceding section to collapse any two sequential FSAs for which all but one of the corresponding states have exactly the same production. For example, the two sequential FSAs in (8a) are collapsed into (8b).

Three and four-state sequential FSAs as in (8b), where at least two of the state-transitions generate more than one morpheme, form the set of *templates* derived from our bootstrapping heuristic. (In work in progress and described elsewhere (see, e.g., Goldsmith and Hu 2004), we use these templates to derive a more general FSA for the morphology of the language.) Each such *template* can be usefully assigned a quantitative score based on the number of letters “saved” by the use of the template to generate the words, in the following sense. The template in (8b) summarizes four words: *aliyesema*, *alimfuata*, *anayesema*, and *anamfuata*. The total string length of these words is 36, while the total number of letters in the strings associated with the transitions in the FSA is $1+4+12 = 17$; we say that the FSA *saves* $36-17 = 19$ letters. In actual practice, the significant templates discovered save on the order of 200 to 5,000 letters, and ranking them by the number of letters saved is a good measure of how significant they are in the overall morphology of the language.

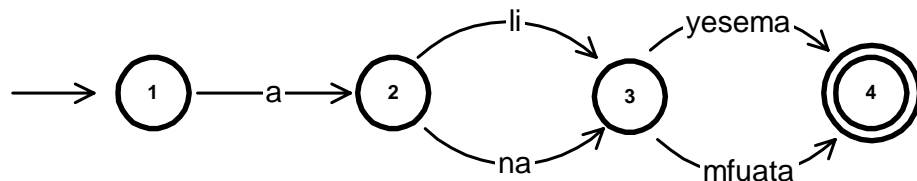
(8)



a.



b.



By this ranking, the top template found in our Swahili corpus of 50,000 running words was one that generated *a* and *wa* (class 1 and 2 subject markers) and followed by 246 correct verb continuations (all of them polymorphemics); the first 6 templates are summarized informally in Figure 1. We note that the third and fourth template can also be collapsed to form a template of the form in (3).

Production of State 1	Production of State 2	Production of State 3
<i>a, wa</i> (sg., pl. animate subject markers)	246 stems	
<i>ku, hu</i> (infinite, habitual markers)	51 stems	
<i>wa</i> (pl. subject marker)	<i>ka, li</i> (tense markers)	25 stems
<i>a</i> (sg. subject marker)	<i>ka, li</i> (tense markers)	29 stems
<i>a</i> (sg. subject marker)	<i>ka, na</i> (tense markers)	28 stems
37 strings	<i>w</i> (passive marker)	<i>a</i>

Figure 1: Top ranked templates

5 Testing and results

In order to support our previous qualitative analyses, we conducted a set of tests based on Swahili, a major Bantu language of East Africa. In the first test, we compared the morpheme boundaries made by the Successor Frequency (SF) heuristic, Predecessor Frequency (PF) heuristic, and SED heuristic against a set of approximately 7,000 Swahili words analyzed by hand.

We applied these heuristics on a corpus of 50,000 running words of Swahili to predict a set of morpheme boundaries in these words. The SED heuristic creates templates, as described above, and in many cases, the same word is analyzed by more than one template. For instance, the word *aliyetokwa* belongs to one template that parses it as *aliyetok + w + a*, while a second template parses it as *aliye + tokwa*. In our first test, we accept *all* the cuts proposed in all templates. In the example considered here, the parse of the word *aliyetokwa* was taken to be *aliye + tok + w + a*.

In our first test, *precision* of a method (SF, PF, SED) is defined as $\frac{\# \text{accurate morpheme boundaries}}{\# \text{total predicted morpheme cuts}}$, and *recall* as the ratio of the number of accurate morpheme cuts to the total number of morphemes in the gold standard. We also report the F-score ($= 2 \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$) to show overall performance. Results are presented in Figure 2, which shows SED scoring twice as well as PF and three times as well as SF.

	Precision	Recall	F-score
SED	0.77	0.57	0.65
SF	0.54	0.14	0.22
PF	0.68	0.20	0.31

Figure 2

We also wanted to see how well the three heuristics performed at finding the most frequent grammatical morphemes of Swahili. In our corpus, the most frequent morphemes were the Class 1 and 2 subject markers (*a*, *wa*), the tense markers *li*, *ka*, *me*, the infinitive marker *ku*, and the morpheme *m*, object marker and noun prefix. Morpheme precision is defined as the ratio of the correct occurrences of a morpheme found (i.e, exact boundaries obtained) to the total occurrences of the morpheme in the predicted parsing, while morpheme recall is the ratio of the correct occurrences of the morpheme found to the number of its occurrences in the gold standard. Results are presented in Figure 3. Again, SED performs significantly better than SF or PF. It is especially striking how poorly PF does in identifying the word-internal Tense Markers (*li*, *ka*, *me*). There is also an effect (only partially seen in this data) whereby PF works better than SF for prefixal systems, and SF better than PF for suffixal systems; but Swahili, like the other Bantu languages, has rich systems of both suffixes and prefixes. Neither SF nor PF are well designed to handle such sequences of word-internal morphemes, thus accounting for their poorer performance in recall in Figure 2.

We therefore explored the extent to which the rank of a template (as determined by the savings of the template, as discussed above) plays a role in determining accuracy of morphological analysis. In Figure 4, we graph precision against recall for the top 10% of the templates, displayed as the leftmost point; for the top 20% of the templates, displayed as the second point from the left; and so on. As we see, accepting more of the templates – moving leftward and up in the graph in Figure 4 – leads to increasing recall and declining precision. The highest ranking templates offer the most reliability, as our model predicted. Figure 5 presents the data from the same experiment, plotting F-score against the number of the top n deciles.

		SED		PF		SF	
Morpheme	Rank	Precision	Recall	Precision	Recall	Precision	Recall
a	1	0.75	0.32	1	0.05	0.0	0.0
wa	2	0.94	0.72	0.97	0.13	0.03	0.01
ku	3	0.96	0.52	0.99	0.52	0.9	0.10
li	4	1.0	0.53	1	0.007	0.5	0.01
ka	5	0.87	0.49	0	0	0	0
m	6	0.62	0.51	1.0	0.10	0	0
na	7	0.96	0.47	1.0	0.04	0	0
me	8	1.0	0.45	0	0	0	0

Figure 3: Precision and Recall of Highest Frequency Morphemes by Heuristics.

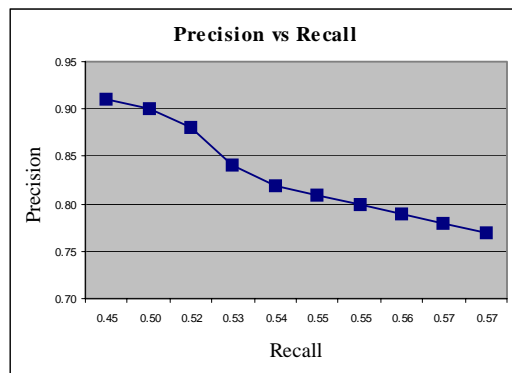


Figure 4 Precision and recall for top deciles of templates

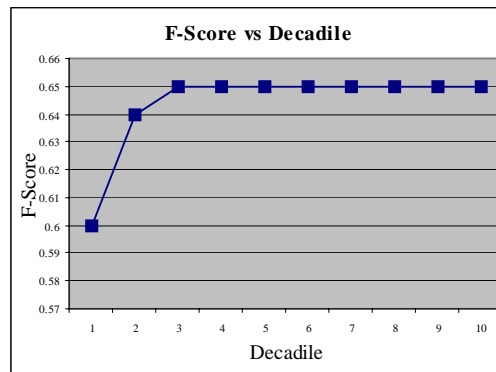


Figure 5 F-scores from selection of top decile of templates

6 Conclusions

In each of the tests we have devised and described in the previous section, the SED-based heuristic is empirically superior to the SF- and PF-based heuristics as a means of identifying morphemes in natural languages with rich morphologies.

The SED-based heuristic that we have described is a rapid method for analyzing data from languages such as Swahili and the other Bantu with a rich morphology, and coming up with a rough characterization of the principal morphemes of the language. By contrast, the methods based on successor frequency and predecessor frequency that have been used in some of the earlier methods embodied in *Linguistica* have not been successful in dealing with such rich morphologies, though we have not demonstrated that in this short paper. Our next step is to integrate this bootstrap heuristic into a large finite state automaton, and employ a Minimum Description Length-based learning method as sketched in Goldsmith and Hu (2004).

References

- GOLDSMITH, J. (2001). Unsupervised Learning of the Morphology of a Natural Language. *Computational Linguistics* 27(2): 153-198.
- GOLDSMITH, J., HU, Y. (2004). From Signatures to Finite State Automata. Midwest Computational Linguistics Colloquium, Bloomington IN.
- HAFER, M. A., WEISS, S. F. (1974). Word segmentation by letter successor varieties. *Information Storage and Retrieval* 10: 371-385.
- HARRIS, Z. (1955). From Phoneme to Morpheme. *Language* 31: 190-222.
- HARRIS, Z. (1967). Morpheme Boundaries within Words: Report on a Computer Test. *Transformations and Discourse Analysis Papers* 73.
- WAGNER, R. A., FISCHER, M. J. (1974). The string-to-string correction problem. *Journal of the Association for Computing Machinery* 21(1): 168-73.