Date of acceptance Grade

Instructor

Cognate Discovery and Alignment in Computational Etymology

Guowei Lv

Helsinki December 29, 2013

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Guowei Lv

Työn nimi — Arbetets titel — Title

Cognate Discovery and Alignment in Computational Etymology

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	December 29, 2013	47 pages + 0 appendices

Tiivistelmä — Referat — Abstract

This master thesis discusses two main tasks of computational etymology. First, finding cognates in multilingual text. Second, finding underlying correspondence rules by aligning cognates.

For the first part, I briefly described two categories of methods in identifying cognates: symbol based and phonetic based. For the second part, I described the Etymon project, which I had been working in. The Etymon project uses a probabilistic method and Minimum Description Length principle to align cognate sets. The objective of this project is to build a model which can automatically find as much information in the cognates as possible without linguistic knowledge as well as find genetic relationship between those languages. I also discussed the experiment that I did to explore the uncertainty in the data source.

ACM Computing Classification System (CCS):
A.1 [Introductory and Survey],
I.7.m [Document and text processing]

Avainsanat — Nyckelord — Keywords
etymology, cognate, minimum description length, dynamic programming

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

In linguistics, *etymology* is the study of the history of words, how words form and meanings change, how languages evolve over time. One of the key concept is *cognates*, *cognates* are words that have a common etymological origin. They often have similar meanings and phonetic features. For instance: night(English), nuit(French), Nacht(German) are cognates, they all mean "night" and belong to the Proto-Indo-European(PIE) language family. We can discover the relationships of different languages by their cognates, it is possible to even construct an hypothetical proto-language for the related languages.

This master thesis discusses two main tasks of computational etymology: finding cognates in multilingual text and finding underlying correspondence rules by aligning cognates.

The identification of cognates can be leveraged to study the relations of different languages. For example, the Etymon project mentioned in this thesis. The methods of discovering cognates roughly fall into two categories: symbol based and phonetic based. And there are also other more complicated methods that integrate semantic information. We will discuss those methods and applications in more detail in Chapter 2.

For finding underlying correspondence rules, I will describe the Etymon project [Yan13], which uses a probabilistic method and Minimum Description Length principle(MDL) to align cognate sets. The cognate sets for Uralic language family are already acquired from some sources, so we don't have to be concerned with the process of finding them here. One objective of this project is to build a model which can automatically find as much information in the cognates as possible without utilizing prior linguistic knowledge. Another objective is to find genetic relationship between languages by building phylogenetic trees based on our model. More details about phylogenetic trees can be found in Chapter 4.3.

The data used in the Etymon project is a dictionary-like cognate set of Uralic language family, which contains more than 30 languages and spoken by approximately 25 million people in eastern and northern Europe and north Asia. The data set contains information of the language name, dialects, explanation of words from the original dictionary [Réd88] and will be described in more detail in Chapter 3.1. In the original dictionary there exist question marks within some entries, which indicate some kind of uncertainty from the authors. I present some experiments to

explore this uncertainty, they can be found in Chapter 4.6.

Based on information theory and Minimum Description Length Principle, the more regularity in the data is found, the more it can be compressed. First we define the procedure of transmitting the data, which is the aligned cognate sets. Then we find the most efficient way to code the data in order to minimize the cost function, which is the code length. The coding methods can be divided into two categories: *non-context* models and *context* models. They will be discussed in more detail in Chapter 3.

One crucial part of the project is the evaluation of our models. This is not an easy and straightforward task. The models give us alignments of the cognate set. Though we can try to tell the goodness of the model by simply looking at the alignments, there are no gold standards or "correct" way of aligning cognates. So we have to conceive other evaluation methods. I describe three methods:

- Using the code length (cost) to represent the power of different models. If the model is smart, then it should be able to get a lower cost, which means more regularity in the data has been found.
- Using the predicting power to evaluate different models. The basic intuition is that given one word in a cognate word pair, our model should be able to impute or "guess" the corresponding words in other languages from scratch. How this imputation method works is described in Chapter 4.4.
- Building phylogenetic trees of languages by using language distances. Phylogenetic tree is often used in bio-informatics to describe the evolutionary relations among biological species. Here if we think of languages as species, then we will get a tree structure of languages of a certain language family, which then can be compared to the tree built by linguists.

2 Cognate Discovery

In general, cognates are words in different languages that share a similar meaning and sound. Cognates can be formed in different situations. If the languages are related, then the cognate words could have the same origin(the proto-language), this is called genetic cognates. In other cases, the languages could be very different, but the cognate can still be identified by the similar pronunciation, this is called

phonetic cognates. This often happens when one language borrows words from the other language, like the Japanese word */hoteru/* is borrowed from the English word *hotel*.

The identification of cognates can be useful in various applications and language research. As in the first part of this thesis discussed, the etymology information could be discovered by aligning the cognates of different languages in the Uralic language family. It employs the heuristic that the relatedness of cognates can represent the relatedness of languages of those cognates. So we consider that the improvement of the quality and quantity of the cognates identified will definitely results in the improvement of the result.

In [Oak00], cognates identification is a crucial part in identifying regular sound changes in related languages. The project aims to find proto-language given its more modern daughter languages.

Other than historical linguistics research, cognates discovery plays an important role in some bitext(text that contains two languages) tasks. Without the intelligent algorithms of cognates recognition, these tasks can be very time consuming and often require a lot of linguistic knowledge about both languages. So it will save both time and human resource if some automatic process could be developed. Here are some brief summaries on projects which benefit from cognate identification, more detailed descriptions of the corresponding methods will be given later.

[SFI92] introduced a way to use cognates as a source of linguistic knowledge to improve the mapping of mutual translations in bitext corpora. The goal is to do alignments, on the sentence level, between the text and its translation of another language. They assumed that translation and cognateness are correlated: there should present more cognateness in translations than random texts. First, they followed the length-based approach developed by [Bro90]. The heuristic is that the length of a sentence and its corresponding translation are related. Though the correctness of this approach is high: 96% of the alignments were found, it tends to make more mistakes when with more complex texts. For example, when the number of sentences in the same paragraph are not the same. And such mistakes could be easily avoided if we incorporate some simple linguistic knowledge such as cognates. For instance, one sentence contains the word *tax* is highly probable to be aligned with the sentence in which contains the word *taxe*.

Injective Map Recognizer (SIMR) by [Mel99] is another bitext mapping algorithm. It formulates the bitext mapping problem in terms of pattern recognition. The bitext

is represented by a rectangular bitext space. The lengths of the axes of this space are the lengths of the two component texts measured by word token, which is character in other words. The objective of the algorithm is to find as complete as possible a set of **True Point Correspondence(TPC)** in the bitext space. TPC can be treated as the point of which the x and y axes represent the mutual translation unit in each text. For example, if a word token on position m on x-axis and a word token on position n on y-axis are mutual translations, then (m, n) is a TPC in bitext space. TPCs can also be on the levels of sentences, paragraphs or even chapters. The cognates identification is used in determining whether two generated candidate points of correspondence are mutual translations.

2.1 Methods

In general, the methods for cognate identification can be grouped into two categories: *orthographic* approaches and *phonetic* approaches. For *orthographic* approaches, the methods are applied on the symbol level and the phonetic features of each symbol are not taken into account. On the other hand, *phonetic* approaches utilize the phonetic characteristics of symbols to measure the relatedness of two words that might be cognates. In the following sub-chapters, we will look at several examples of methods in both categories.

2.1.1 The orthographic approaches

[SFI92] introduced a relatively easy way to find cognates from two sentences of different languages. The definition for cognates in this task is a little bit different as usual. Tokens *t1* and *t2* are considered to be cognates under the following conditions:

1. *t1* and *t2* are identical and both contain at least one digit.
2. *t1* and *t2* contain only letters, and at least four letters long. The first four letters of each are identical.
3. *t1* and *t2* are identical and both are punctuation.

The first and third conditions are for recognizing non-character parts in the sentence, for example if *50mm* appears in one sentence, then the corresponding translation will also have *50mm*. For regular words, the algorithm only considers the first four characters.

As one can see, this algorithm is based on a very simple heuristic. The advantage is that it's easy and efficient. The disadvantage is that many cognates could be missed. For example, *government* in English and *gouvernement* in French are cognates, but the algorithm fails to recognize them because the first four characters are not the same.

Even though the algorithm is extremely naive, it gives rather good results for discovering sentences that are mutual translations. For instance, the *cognateness* γ is 0.3 for mutual translations and 0.09 for random sentences. The *cognateness* γ is the measurement of the relatedness of two sentences in the sense of cognates. Defined as:

$$\gamma = \frac{c}{(n+m)/2}$$

where c is the number of cognates discovered, n and m are the number of tokens in the two sentences.

Dice's coefficient is another way to discover cognates in sentences. In general, it is a similarity measurement over sets, defined as:

$$s = \frac{2|X \cap Y|}{|X| + |Y|}$$

where X, and Y are two sets. Dice's coefficient is named after Lee Raymond Dice, and originally introduced to measure the association between species. When applied to measure string similarity, *bigram* is often used. It was first adopted by [McO96] for cognates recognition. The Dice coefficient of string x and y can be calculated as follows:

$$D(x, y) = \frac{2|Set_{bigram}(x) \cap Set_{bigram}(y)|}{|Set_{bigram}(x)| + |Set_{bigram}(y)|}$$

For example, the English word *star* and Romanian word *stea* are cognates. The bigram sets of the two words are: *{st, ta, ar}* and *{st, te, ea}*. So the $D(star, stea) = 2 \times 1/(3+3) \approx 0.33$.

[Mel99] decides whether two tokens are cognates or mutual translations by thresholding the Longest Common Subsequence Ratio (LCSR). The LCSR is calculated by the following formula:

$$LCSR(A,B) = \frac{length(LCS(A,B))}{max\{length(A), length(B)\}}$$

where LCS(A, B) stands for the longest common sequence for A and B, the sequence are not necessarily contiguous. For example, the longest common sequence for words *night* and *nacht* is *n-h-t*. Therefore the LCSR(*night*, *nacht*) = 3/5=0.6. The time complexity for calculating LCSR is $O(n^2)$ for simple dynamic programming algorithm (Bellman 1957) and can be improved to $O(nloglogn)$ by other more complicated algorithm(Hunt and Szymanski 1977).

2.1.2 The phonetic approaches

[Kes95] discussed two methods to measure distance between phonetic strings by computing *Levenshtein distance*. *Levenshtein distance* is also called *edit distance* and originally defined as the number of operations that can transform one word into another. The operations include *insertion*, *deletion* and *substitution*. Furthermore, each operation can be assigned with different costs, so the Levenshtein distance can also be defined using the cost of all the operations instead of the number of the operations, and less cost means more similarity.

The first approach introduced is called *phone string comparison*. In it, all operations cost 1 unit. But they argue that it is not fair to assign each phonetic operation the same cost. For example the substitution between [a] and [A] is not as distinctive as the substitution between [a] and [t]. Thus, they developed a more complicated *feature string comparison* method. It used twelve phonetic features to represent different phones. Each feature has several values, represented by discrete ordinal numbers. In this way, for calculating the *Levenshtein distance* between two phonetic feature strings, instead of using the 1 unit cost for all the operations, they use the averaged difference between the twelve features for every phone. But when comparing those methods with the traditional base method of plotting isoglosses, it showed that the *phone string comparison* method outperformed the more complex *feature string comparison*. The author argued that this does not mean that the phone string comparison method using the naive 1-unit cost is superior to the other more complex and sensitive feature string comparison method. And the latter may work better if the weights of the features are assigned differently.

Similar to [Kes95], [NeH97] also used *Levenshtein distance* based methods to measure the dialect distance. The methods are either based on atomic characters or

feature vectors. For the feature vector based methods, they used a vector of 14 features to represent phonetic symbol. Then, they experimented with three methods to measure the phonetic distance. First is *Manhattan Distance* (also called "city block" distance). It just takes the sum of the differences of all 14 features, so the distance of symbol $\mathbf{x}$ and $\mathbf{y}$ is calculated as:

$$Distance(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n=14} |\mathbf{x}_i - \mathbf{y}_i|$$

The second method they tried was *Euclidean Distance*. As the original definition, it takes the square root of the sum of the squared feature differences:

$$Distance(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n=14} (\mathbf{x}_i - \mathbf{y}_i)^2}$$

Third, they used *Pearson Correlation*, and the distance was described as follows:

$$Distance(\mathbf{x}, \mathbf{y}) = 1 - \frac{1}{n-1} \sum_{i=1}^{n} \left(\frac{\mathbf{x}_i - \overline{\mathbf{x}}}{s_{\mathbf{x}}} \right) \left(\frac{\mathbf{y}_i - \overline{\mathbf{y}}}{s_{\mathbf{y}}} \right)$$

where $s_{\mathbf{x}}$ and $s_{\mathbf{y}}$ are standard deviations.

As for evaluation for these methods, they compared the results to the well-established scholarship in dialectology. They claimed that all the methods perform quite well, and the top performing method was based on feature vectors and *Manhattan Distance*.

[Kon02] developed a more complex system called COGIT, which is a similarity based cognate identification system. It basically consists of two parts. The first part is ALINE, which is an algorithm to measure the similarity of phonetic features. The second part is a procedure to find semantic similarity from glosses that comes with the word lists. I will describe the ALINE algorithm in more detail first.

In general, ALINE is a cognate alignment algorithm, which employs dynamic programming for finding the optimal alignment, and the metric for comparing phonetic segments is based on phonetic similarity and multivalued features. The key components of ALINE are described below.

The first thing we should elaborate is dynamic programming. Here I will give the pseudo code, since it will be referred to and modified later.

D(0, 0) := 0
for $i \leftarrow 1$ **to** n **do**
  $D(i, 0) \leftarrow D(i-1, 0) + \delta(a_i, -)$;
end
for $j \leftarrow 1$ **to** m **do**
  $D(0, j) \leftarrow D(0, j-1) + \delta(-, b_j)$;
end
for $i \leftarrow 1$ **to** n **do**
  for $j \leftarrow 1$ **to** m **do**

$$D(i,j) \leftarrow min \begin{cases} D(i-1, j-1) + \delta(a_i, b_j) \\ D(i-1, j) + \delta(a_i, -) \\ D(i, j-1) + \delta(-, b_j) \end{cases}$$

  end
end

Algorithm 1: Dynamic programming for word pair alignment

The mission of this algorithm is to fill in a $(n+1) \times (m+1)$ cost matrix, where n and m are the lengths of the two words a and b. $D(i, j)$ refers to the minimal cost up until the i^{th} character of a and j^{th} character of b that have already been aligned. $\delta(a_i, b_j)$ denotes the cost of aligning a_i to b_j (substitution). And similarly, $\delta(a_i, -)$ and $\delta(-, b_j)$ represents the costs of aligning a_i to *nothing* (deletion) and aligning *nothing* to b_j (insertion) respectively. Here we assume the insertion/deletion cost to be 1 and substitution cost to be 0 for identical segments, and 2 otherwise.

Using the algorithm above, we will find the best alignment of two words in terms of distance. Alternatively, the alignment can also be performed by using similarity measurement. Similar to the distance, the similarity score of two words can be defined as the sum of all the similarity scores of each aligned character. Usually, the similar alignment will get large positive score and the dissimilar alignment will get large negative score, and the deletions or insertions will be assigned small negative scores. For the dynamic programming algorithm to work with similarity measurement, we only need to change the min to max, and the δ now stands for similarity score.

The reason why similarity measurement is used instead of distance is to perform *local alignment*. *Local alignment* is preferred when the sub-strings of two words share high similarity , while the complex affix may harm the performance of aligning the words as a whole. The goal of *local alignment* is to find the optimal sub-strings alignment of

the original words that have the highest similarity score, and leave out the unrelated affixes. This goal can not be achieved by minimizing the distance because the lowest distance is always 0, when the two parts in the alignment are identical. In this case, by minimizing the distance, we will always end up with relatively short identical sub-strings alignment or empty alignment.

Local alignment can be easily performed by making some straightforward adaptations to the similarity-based dynamic programming algorithm. The pseudo code is as follows.

S(0, 0) := 0
for $i \leftarrow 1$ **to** n **do**
    $S(i, 0) \leftarrow 0$;
end
for $j \leftarrow 1$ **to** m **do**
    $S(0, j) \leftarrow 0$;
end
for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** m **do**

$$S(i,j) \leftarrow max \begin{cases} S(i-1, j-1) + \delta(a_i, b_j) \\ S(i-1, j) + \delta(a_i, -) \\ S(i, j-1) + \delta(-, b_j) \\ 0 \end{cases}$$

    end
end

Algorithm 2: Dynamic programming for local alignment with similarity measurement

$S(i, j)$ now contains the largest similarity score of the affixes of the sub-strings of a and b up until i^{th} and j^{th} character, respectively. The first row and column of the similarity matrix is initialized as zeros. δ now represents the similarity of alignments instead of distance cost. They are also predefined values.

For example, āpakosīs in Cree and wāpikonōha in Fox both mean "mouse". The similarity matrix generated by applying dynamic programming is illustrated in Figure 1.

For simplicity, assume that the similarity score for substitution is -5, for aligning

		0	1	2	3	4	5	6	7	8
			ā	p	a	k	o	s	ī	s
0		0	0	0	0	0	0	0	0	0
1	w	0	0	0	0	0	0	0	0	0
2	ā	0	5	4	3	2	1	0	0	0
3	p	0	4	10	9	8	7	6	5	4
4	i	0	3	9	8	7	6	5	4	3
5	k	0	2	8	7	13	12	11	10	9
6	o	0	1	7	6	12	18	17	16	15
7	n	0	0	6	5	11	17	16	15	14
8	ō	0	0	5	4	10	16	15	14	13
9	h	0	0	4	3	9	15	14	13	12
10	a	0	0	3	2	8	14	13	12	11

Figure 1: Similarity score matrix for local alignment

identical characters is 5, for insertion/deletion is -1.

To find the highest scoring alignment of sub-strings, we need to first find the highest score in the matrix, which is 18. Then trace back until we encounter a 0 entry. So the optimal sub-strings alignment is:

w || ā p a . k o || nōha
|| | | | | | | ||
|| ā p . i k o || sīs

So in this way, we successfully find the common part of the two words without the unrelated affixes.

Another adaptation of the dynamic programming adopted by ALINE is that one extra pair of edit operation *compression/expansion* was added. So the set of edit operations becomes *substitution insertion/deletion* and *compression/expansion*. Compression here means two contiguous characters of one string corresponds to one single character in the other string, while *expansion* is the other way around. The

f	a	kt
|	|	|
-	e	č

Table 1: An example of cognates alignment using edit operation compression/expansion

Manner	
[stop]	1.0
[affricate]	0.9
[fricative]	0.8
[approximant]	0.6
[high vowel]	0.4
[mid vowel]	0.2
[low vowel]	0.0

Table 2: Values for feature *Manner*

benefit of this addition is that it can help when the correspondence is more complex. Take Latin *factum* and Spanish *hecho* for example, the sound [č] is derived from the combination of [k] and [t], so with *compression/expansion*, the correct alignment is shown in Table 1.

ALINE employs the multivalued features with salience coefficients in its feature alignment procedure. ALINE uses the multivalued feature system developed by [Lad95]. All features' values are real numbers in the range of [0,1]. Table 2 shows the values of feature *Manner*. For each feature, ALINE assigns a *salience* value indicating its importance. It is argued in the paper that the salience values should be tuned carefully to gain a reasonable result, especially for features [place] and [manner]. The default salience values adopted by ALINE is shown in Table 3, and can be modified by hand to gain better result for different languages.

The ALINE algorithm combines those techniques above. In dynamic programming, the similarity score is used instead of distance score, multivalued feature with salience coefficients and local alignments are also used to improve the performance.

2.1.3 The semantic approaches

ALINE is based on phonetic similarity. The GOGIT algorithm[Kon02] also uses semantic similarity to assist the cognate discovery. The input of the algorithm are

Syllabic	5	Place	40
Voice	10	Nasal	10
Lateral	10	Aspirated	5
High	5	Back	5
Manner	50	Retroflex	10
Long	1	Round	5

Table 3: Features and their salience values

lists of words with their corresponding glosses, which is a short text explaining the meaning of the word. The basic idea is to discover the "cognateness" of two words by analysing their glosses. Because of the characteristics of glosses, such synonymy and semantics changes, simply comparing the glosses and seeking for common words might not work directly.

Some of the problems can be solved by applying some intuitive preprocessing to the glosses data. For example, additional modifiers, like "*small* stone/stone" and "*my* finger/finger" can be solved by specifying a list of stop-words, such as *a, such, many, kind of,* etc. But there exist some more complex situations. For example, synonymy. It is very difficult for the algorithm to treat "grave" and "tomb" as identical just by preprocessing the data. So to solve this kind of problem, the GOGIT integrate the external lexical resource called WordNet [Fel98].

WordNet is a free lexical database of English language. Words that denote the same concept are grouped together, these groups are called synonymy sets (synsets). Synsets are linked by several "conceptual relations", some of the relations for nouns are shown in Table 4. For example, hypernymy is the most frequently encountered relation among the synsets. It links more general synsets to increasingly specific ones, as we can guess from the name "IS-A". Note that hypernymy relation is transitive, if {bed} -> {furniture} and {bunkbed} -> {bed}, then {bunkbed} -> {furniture}, in this case the links can be traversed from top to bottom.

The process of computing the semantic similarity score with the assist of WordNet is as follows. The input to GOGIT are two word lists *ListA* and *ListB*, and each word has an English gloss attached to it. The first step is to preprocess the data, including removing all the stop words in glosses and finding keywords in the glosses by applying a part-of-speech tagger. After that, the gloss for each word in *ListA* and *ListB* contains only noun keywords. The second step is to generate lists of synonyms, hyponyms and meronyms separately for each keyword in the glosses.

Relation	Short name	Example
hypernymy	IS-A	{bed} -> {furniture}
meronymy	PART-WHOLE	{leg} -> {chair}
holonymy	HAS-A	{tree} -> {branch}

Table 4: Examples of relations between noun synsets

Rank	Similarity level	Score
1	gloss identity	1.00
2	gloss synonymy	0.70
3	keyword identity	0.50
4	gloss hypernymy	0.50
5	keyword synonymy	0.35
6	keyword hypernymy	0.25
7	gloss meronymy	0.10
8	keyword meronymy	0.05
9	none detected	0.00

Table 5: Semantic similarity levels

The next step is to calculate the semantic similarity scores based on a 9-point scale of semantic similarity, which is shown in Table 5. Note that the program checks the levels in the order of the rank, and if it detects one level, it will cease and use that score. The names of the similarity levels are quite self-explanatory. For instance, "gloss identity" means the whole glosses are the same for the two entries, "keyword hypernymy" means that a keyword in one gloss is a hypernymy of a keyword in the other gloss. The final similarity score of the two word entries is a linear combination of *semantic similarity score* and *phonetic similarity score*.

As for the performance of the WordNet in real application, the author claims that it is very limited, the similarity scores that are affected by adopting WordNet account for less than 10% of the cognate pairs. One reason is that the data set used is based on a single project, that leads to lack of diversity. As a result, many cognates have almost identical glosses and that makes WordNet unnecessary. Another problem is that some of the keywords extracted from the glosses cannot be recognized by WordNet even after preprocessing, like "spawner", "spotfish". This also makes it difficult to use WordNet efficiently.

3 Cognate Alignment

In the previous chapters, I gave a brief summary on methods to discover cognates in multi-lingual context. In this chapter, I will discuss how to find the relations of languages by aligning their cognate sets in Etymon project.

3.1 Data Description

In the Etymon project, the data source used is StarLing Database of Uralic Languages from "Tower of Babel" web-based project [Sta13], which is an international etymological database project of historical and comparative linguistics. The Uralic etymology database is heavily based on K. Rédei's dictionary [Réd88]. Figure 2 illustrates how cognates are stored in the xml file.

```
<record id="31">
  <field name="NUMBER">31</field>
  <field name="PROTO">*aśke (˜ -lV), *ać̀ke ( ˜ -lV)</field>
  <field name="PRNUM">1273</field>
  <field name="MEANING">step</field>
  <field name="GERMMEAN">Schritt</field>
  <field name="FIN">askel (gen. askelen) “Schritt”, ? astu- “steigen, treten, gehen”</field>
  <field name="EST">askelda-, aselda- “sich mit etwas beschäftigen”, ? astu- “schreiten, treten, steigen”</field>
  <field name="MRD">aśkiĺa-, eśkiĺa-, iśkila- (E), aśkoĺa- (M) “schreiten”</field>
  <field name="MAR">aškâl (KB), oškâl- (B) “Schritt”, aškâla- (J), oškâla- (U) “schreiten, Schritte machen”, aškeδä-
    (KB) “treten, gehen, steigen”</field>
  <field name="UDM">ućkįl- “Schritt”</field>
  <field name="KOM">vośkol (S), ośkęl, ośkęv (P), u\H.\hśkøl (PO) “Schritt”, voślal- (S), oślal- (P) “schreiten”</
    field>
  <field name="MAN">ūsəl (KO), ūsil (So.) “Schritt”, ūssul- (N) “langsam schreiten”, ūsuwl- (N) “überschreiten”</
    field>
  <field name="SLK">aasel- (Ta.) “überschreiten”</field>
  <field name="QUALITY">!  U</field>
  <field name="FU">FiEs Md Mr Ud Ko Vg</field>
  <field name="SAMOYED">Sk</field>
  <field name="SAMM2">(FU?) *as'ki/ali</field>
  <field name="HELIMSKI">Hu oson: vgl. orvos</field>
  <field name="LITERATURE">FUV; SKES; КЭСКЯ; MUSz. 858; SzófSz.; Paas. Beitr. 241; Stein.Fgr.Vok. 36; Coll.CompGr. 95,
    153, 405; Munkácsi B. Árja és kaukázusi elemek a finn-magyar nyelvekben 224</field>
  <field name="UEW___SW">19</field>
</record>
```

Figure 2: data entry in xml file.

In the xml file, each word is represented by a *record* tag, which contains several fields. The PROTO field is the general "prototype" of this word entry, it may not exist in real languages. The MEANING field is the English meaning of the PROTO word, in this example "step". Some other fields stand for various forms of the word in different languages. For example, FIN is Finnish, EST is Estonian, MRD is Mordovian and so forth. Each language field contains word forms of different dialects (if exist) in that language, shown within parentheses. For instance, E(Erzya) and M(Moksha)

are two dialects of Mordovian. The UEW__SW field is the page number for this word in the original Rédei dictionary.

There exist other etymology data sources that could be alternatively used in the project in the future. For example, the digital Finnish etymology dictionary SSA-Suomen Sanojen Alkuperä (the origin of Finnish words)[ItK00], which is centered on Finnish vocabulary.

3.2 Information Theory

Information theory was first established by Claude E. Shannon in 1948 [Sha48], involving quantification of information. It is based on probability theory and statistics to find limits of signal processing.

One important quantity in information theory is *entropy*. Intuitively, *entropy* describes the "unexpectedness" or "uncertainty" of a random variable. To explain it from the point of view of probability, let's take the classic example of tossing a coin and let e be the event of obtaining HEAD or TAIL. If the coin is fair then the probability of head and tail are equal so it is hard to predict the outcome, or in other words, the "uncertainty" is high. In this case, the *entropy* is high. Otherwise, if the coin is biased, then it is easier to predict the outcome and the "uncertainty" is lower, then the *entropy* is lower. This is shown in Figure 3.

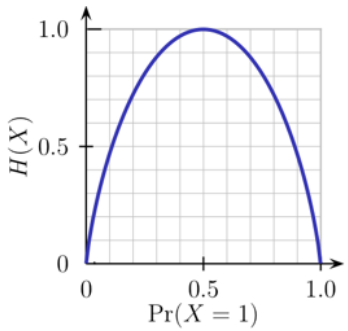

Figure 3: Entropy of tossing a coin. The entropy is highest when P(X=1)=0.5

Here is a formal definition. Given a discrete random variable X with values $\chi = \{x_1, x_2, ..., x_n\}$. Let P_X be the mass probability function. Then the information or "surpriseness" of value x can be measured by Equation 1.

$$I_X(x) = \log_2 \frac{1}{P_X(x)} \tag{1}$$

Entropy H is the expected amount of "surpriseness" of all the values:

$$H = E\left[I_X(X)\right] = \sum_{x \in \chi} P_X(x) \log_2 \frac{1}{P_X(x)} \tag{2}$$

Data compression and communication are two fundamental concepts in information theory. The basic setting is that the *sender* wants to send the data to the *receiver* over a channel. The data needs to be encoded before sending, using as few bits as possible, and the receiver should be able to decode and recover completely the original data after receiving it.

Shannon's *Source Coding Theorem* established a limit to data compression, it states that the number of bits needed to code the uncertain variable is limited to its entropy. One intuitive principle of compressing the source data is to reduce the redundancy and try to shorten the code length as much as possible. For example, use shorter code words for more frequent data sequences and longer code words for less frequent data sequences.

3.3 The MDL Principle

The minimum description length (MDL) principle is a inductive inference method, the basic idea is to compress data using some description method or "code", and the more *regularity* in the data we discover, the more we can compress the data using the *regularity* found[Grü07], and thus the more we have learned from the data.

To explain what is a description method, consider a data sequence $D = (x_1, x_2, x_3, ..., x_n)$ where x_i comes from some observation space X. Without losing generality, the data sequence will be described as binary bit strings. Therefore, a *description method* is a one-to-many relation from the observation space to the set of binary strings of arbitrary length.

Ray Solomonoff introduced the use of a *universal computer language* as a description method[Sol64]. For every data sequence D, we are looking for a piece of computer language code that prints out D and then halts. In that case, the description method can be defined as the *shortest* program code that prints D and then halts. We call

such program the *optimal hypothesis* for D. And the length of the shortest program is defined as the *Kolmogorov complexity*. The more the Kolmogorov complexity of data D, the less regular, or the more random it is. However, there are problems when applying Kolmogorov complexity to practical problems, since the Kolmogorov complexity is not computable. It is proved that there exists no computer program that for any input data sequence D, gives the shortest program that prints D and halt [LiV08].

In general, the basic idea of the MDL principle is to solve the problem of applying Kolmogrov complexity in practice by using description methods that are less expressive than general-purpose computer languages. Such a description method should be general enough to allow us to compress as many "regular" data sequences, and at the same time restrictive enough to allow the computation of shortest description length of any given data sequence D.

Two-part code is a coding method which adopts the MDL principle. It states that the best hypothesis H to explain the data D is the one which minimizes the sum $L(H)+L(D|H)$, where $L(H)$ is the code length of the description of the hypothesis and $L(D|H)$ is the code length of the data given the hypothesis H.

3.4 Aligning Cognates

As described in Chapter 3.1, each record entry in the database contains several Uralic languages. And the alignments are made between different languages within the same record entry. In this thesis, we only consider the case, where two words get aligned at a time and the symbols are aligned in a 1-1 way(pairwise 1-1 alignment). In the pairwise 1-1 alignment, we are aiming at finding the symbol correspondence within the two words. Though alignment doesn't have an order, let's call the first word *source* word and the second word *target* word for convenience. The source word $\mathbf{s}$ and target word $\mathbf{t}$ can be defined as vectors consist of symbols from their alphabets. The symbol-wise alignment can be denoted by $(s:t) \in S \times T$ where s is a symbol from *source alphabet* S and t is a symbol from *target alphabet* T. The length of $\mathbf{s}$ and $\mathbf{t}$ are denoted as $|\mathbf{s}|$ and $|\mathbf{t}|$ respectively. Since the length of the words are not always equal, we include a special empty symbol "." in both alphabets. This will allow us to do *insertions* and *deletions* so that we can align words with different length. The extended alphabets are denoted as S. and T.. Below shows two possible alignments of the word pair *aivo* - *aju*(meaning "brain" in Finnish and Estonian).

(1)
a	i	v	o
|	|	|	|
a	j	u	.

(2)
a	i	v	o
|	|	|	|
a	.	j	u

The first alignment contains symbol pairs: (a, a), (i, j), (v, u), (o, .), and the second alignment contains: (a, a), (i, .), (v, j), (o, u).

3.5 Baseline Model

Our goal is to discover the underlying rules of symbol correspondence by aligning cognates to each other. Given the data $D = (\mathbf{s_1}, \mathbf{t_1}), (\mathbf{s_2}, \mathbf{t_2}), ..., (\mathbf{s_N}, \mathbf{t_N})$, which contains N word pairs, we need to find the "best" alignment for each pair. According to the Minimum Description Length principle, the more we can compress and transmit the "alignments", the more "rules" or "regularity" are found. The "rules" in the sense of cognates aligning can be something like, for example, similar looking symbols in two sister languages are more likely to be aligned. The problem then transforms into finding a set of symbol corresponding rules that can minimize the code length of a certain coding scheme. And in order for the code to be decodable, we add a special "♯" symbol at the end of each word as the word boundary. We use a *global matrix* M to keep track of the alignments of the entire cognate set D of two languages. $c(s_i, t_i)$ simply represents the count of corresponding symbol pair s_i and t_i in all word alignments. Figure 4 shows how the global count matrix looks. Obviously, since there are N alignments, $c(\#, \#) = N$.

	.	t_1	t_2	...	$t_{\|T\|}$	#
.	-	$c(s_1, t_1)$				-
s_1						-
s_2	$c(s_2, .)$		$c(s_2, t_2)$			-
...						-
$s_{\|S\|}$					$c(s_{\|S\|}, t_{\|T\|})$	-
#	-	-	-	-	-	N

Figure 4: Global count matrix.

In this setting, what we really need to transmit are a sequence of events of type e from the event type space $E = S. \times T. \cup (\sharp : \sharp)$. The *event* is an action of aligning a symbol from *source* word to a symbol of *target* word. The *type* of the *event* is defined by the representation of the symbols in their corresponding alphabet. We use *prequential* coding [Daw84] as the coding scheme.

The procedure of *prequential* coding can be comprehended as a "coding game" played by a *sender* and a *receiver*. The mission is that the *sender* can transmit the data, in the form of aligned word pairs, to the *receiver* and the *receiver* can uniquely and completely decipher it. The "$\sharp$" symbol added at the end of each word assists this procedure to separate the words from each other. The process is to send the symbol pair alignments one by one, from the whole set of word pairs D. Both participants know the structure of the data, the alphabets, by default.

Both the *sender* and the *receiver* keep track of the number of different type of events sent/received by far, and then update the probability distribution of different event types after transmission of every single event.

Let N be the number of event types ($N = |E|$). At the initial point, the probabilities of all the event types are set to $1/N$. The number of events of type e_i sent is denoted by $c(e_i)$. As the events are sent, the probability distribution will be updated at the same time. Table 6 shows how the probability distribution is updated. Note the order of the events does not matter.

At last, the probability of sending the whole data set can be computed as the product of the probability of each event, as shown in the following equation:

$$P(D) = \frac{c(e_1)! \cdot c(e_2)! \ldots \cdot c(e_N)!}{\frac{(\sum_{i=1}^{N} c(e_i)+N-1)!}{N-1}} \tag{3}$$

Since the total code length $L(D)$ can be derived from the probability as follows:

$$L(D) = -log_2 P(D) \tag{4}$$

Factorials can be replaced by gamma function, $\Gamma(n) = (n-1)!$, where $n \in \mathbb{Z}^+$. So the total code length can be written as:

Table 6: The updates of the probability distribution as events being sent

	$p(e_1)$	$p(e_2)$	...	$p(e_N)$
Initial	$\frac{1}{N}$	$\frac{1}{N}$	...	$\frac{1}{N}$
After the first event of type e_1 came	$\frac{2}{N+1}$	$\frac{1}{N+1}$	...	$\frac{1}{N+1}$
After the second event of type e_1 came	$\frac{3}{N+2}$	$\frac{1}{N+2}$	...	$\frac{1}{N+2}$
After the last event of type e_1 came	$\frac{c(e_1)+1}{N+c(e_1)}$	$\frac{1}{N+c(e_1)}$	...	$\frac{1}{N+c(e_1)}$
After the first event of type e_2 came	$\frac{c(e_1)+1}{N+c(e_1)+1}$	$\frac{2}{N+c(e_1)+1}$	...	$\frac{1}{N+c(e_1)+1}$
After the second event of type e_2 came	$\frac{c(e_1)+1}{N+c(e_1)+2}$	$\frac{3}{N+c(e_1)+2}$	...	$\frac{1}{N+c(e_1)+2}$
After the last event of type e_2 came	$\frac{c(e_1)+1}{N+c(e_1)+c(e_2)}$	$\frac{c(e_2)+1}{N+c(e_1)+c(e_2)}$	...	$\frac{1}{N+c(e_1)+c(e_2)}$
...	...	...	...	...
After the last event of type e_N came	$\frac{c(e_i)+1}{N+\sum_{i=1}^{N} c(e_i)}$	$\frac{c(e_2)+1}{N+\sum_{i=1}^{N} c(e_i)}$	...	$\frac{c(e_N)+1}{N+\sum_{i=1}^{N} c(e_i)}$

$$L_{base}(D) = -\sum_{e' \in E} \log \Gamma(c(e') + \alpha(e')) + \sum_{e' \in E} \log \Gamma(\alpha(e')) + \log \Gamma \left[\sum_{e' \in E} (c(e') + \alpha(e')) \right] - \log \Gamma \left[\sum_{e' \in E} \alpha(e') \right] \quad (5)$$

All the alignment counts are stored in the *global matrix M*, where $M(i, j) = c(i, j)$. We use uniform prior in all of our models, so $\alpha(e) = 1, e \in E$.

Here is the outline of the algorithm for baseline model: First of all, we randomly align all the word pairs, in this way, the global count matrix M is also randomly initialized by just count the number of each symbol alignment in the whole corpus. Then we repeatedly loop through the whole corpus, realign each word pair at a time using dynamic programming combined with simulated annealing (see Chapter 3.6), until convergence of the cost function (Equation 5). Actually, there are 3 steps in the re-alignment of one word pair: First, subtract the word's contribution to the global matrix. In other words, for each symbol alignment in the word pair, find the

corresponding cell in the global matrix and reduce the number by 1. Second, realign the word pair using dynamic programming. Third, register the new alignment back to the global matrix by adding 1 to the corresponding cells.

3.6 Dynamic Programming with Simulated Annealing

The basic idea of *dynamic programming* is to divide a complex problem into simpler subproblems. To solve the complex problem, we first solve the simpler subproblems and save the results, then combine them to get the final solution. When many of the subproblems are the same, dynamic programming will reduce the computation by only solve each of the subproblems once. In our model, we use dynamic programming to find the best alignment of a word pair, which has the lowest cost.

Now we illustrate the algorithm in detail. We first construct a matrix for a word pair $(\boldsymbol{s}, \boldsymbol{t})$, where source word $\boldsymbol{s} = [s_1, s_2, ..., s_n]$ and target word $\boldsymbol{t} = [t_1, t_2, ..., t_n]$. We fill this matrix from left to right, top to bottom. Each path from top-left to bottom-right in this matrix corresponds to a unique way to align the word pair. Take the word pair $aivo - aju$ as an example. Figure 5 shows 2 different alignments.

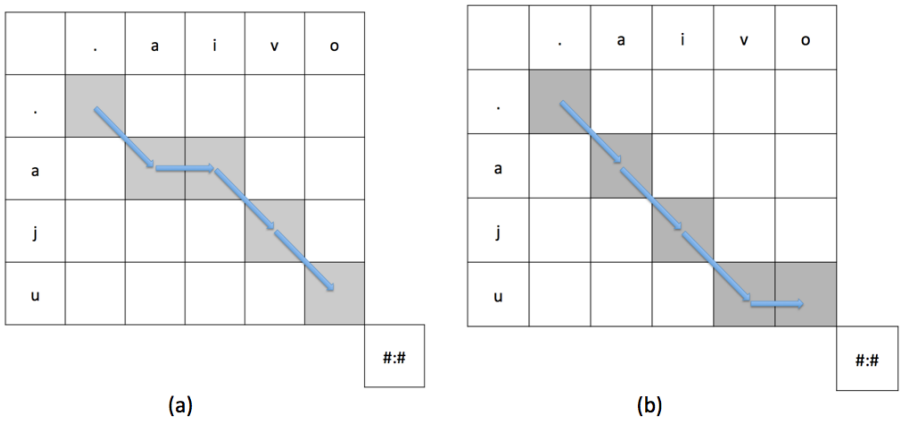

Figure 5: Two different paths of alignment in dynamic programming matrix. The alignment indicated by (a) is: [(a -a), (i - .), (v - j), (o - u)] . The alignment indicated by (b) is: [(a - a), (i - j), (v - u), (o - .)].

Each cell (s_i, t_i) means the symbols up to (s_i, t_i) in both words have been aligned, and in it stores the cost $C(s_i, t_i)$ of the most probable path so far. The cost at the

starting point (top-left) is 0, and the optimal cost of the whole alignment is stored in the bottom-right cell. Our goal is to find optimal path with the lowest cost.

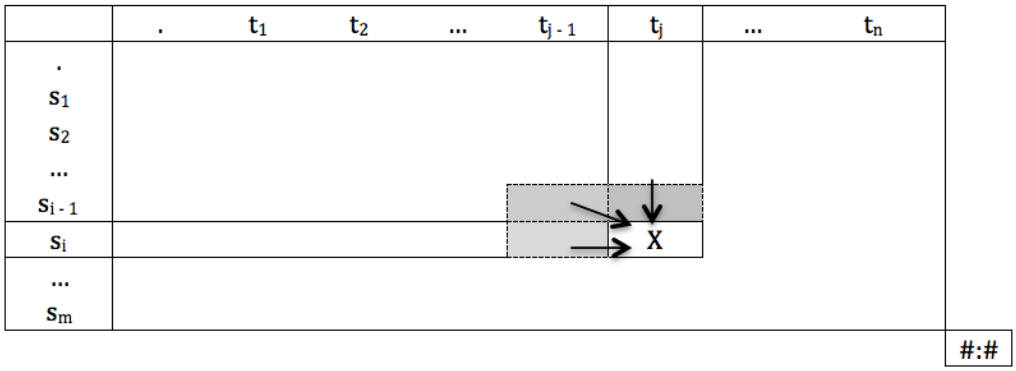

Figure 6: Dynamic programming matrix.

For a certain cell X in the dynamic programming matrix, there are three ways that can lead to it, illustrated in Figure 5. So the cost of the cell X is calculated as follows:

$$C(s_i, t_j) = min \begin{cases} C(s_i, t_{j-1}) & +L(. : t_j) \\ C(s_{i-1}, t_j) & +L(s_i : .) \\ C(s_{i-1}, t_{j-1}) & +L(s_i : t_j) \end{cases} \tag{6}$$

The cost of the three preceding cells have already been calculated by dynamic programming at this point. The three arrows stand for three events: aligning s_i to ., aligning s_i to t_j and aligning . to t_j. For any observed event ϵ of type e, $L(\epsilon)$ is the cost of ϵ. Then $L(\epsilon)$ can be calculated by the change of the total code length before and after the event ϵ is observed, shown in Equation 7, where $\mathcal{E}$ is the set of event *instances* so far.

$$L(\epsilon) = \Delta L(\epsilon) = L(\mathcal{E} \cup \{\epsilon\}) - L(\mathcal{E}) \tag{7}$$

The most probable path is equivalent to the lowest cost path, based on information theory, the probability of event ϵ can be converted to the cost of observing event instance ϵ via Equation 8. Combining Equation 5 and Equation 8 gives Equation 9, so that we can compute $P(\epsilon)$ explicitly.

$$P(\epsilon) = 2^{-\Delta L(\epsilon)} = \frac{2^{-L(\mathcal{E}\cup\{\epsilon\})}}{2^{-L\{\mathcal{E}\}}} \tag{8}$$

$$P(\epsilon) = \frac{c(e)+1}{\sum_{e'\in E} c(e') + |\mathcal{E}|} \tag{9}$$

Our first try is to use greedy search to do the optimization of the target cost function. In greedy search, we assume that local optimization will lead to a global optimum. In the dynamic programming phase, given a state, the next state is always following the most probable path. In this way, we keep iterating over the whole corpus until the cost function converges.

Although the greedy search has a fast converging time, the result usually falls into some local optima with a relatively high cost. To solve this problem, simulated annealing has been integrated.

Simulated annealing[KGV83] is a probabilistic method for finding the approximation of the global optimum based on a cost function, while there exists multiple local optima. It is an analogous method of the annealing process in metallurgy, where metals are slowly cooled down from a high temperature in a well controlled manner, in order to increase the size of its crystals and reduce their defects.

The basic idea is that instead of choosing the best solution for each step, we factor in some "randomness" to prevent us from falling into local optima. There are several terms and parameters in simulated annealing: $T(t)$ is the temperature at time t. $T(0)$ is called the initial temperature, we use $T(0) = 50$ or $T(0) = 100$ in this project. The cooling schedule is defined as a function $T(t) = \alpha T(t-1)$, where α is called cooling parameter. We use $0.99 \leq \alpha \leq 0.995$. The smaller the cooling parameter, the faster the cooling process will be. When the temperature T is large, the choices we make are almost random, and as the temperature slowly goes down, we decrease the randomness and thus increase the probability of making the best choice. In the dynamic programming process, to fill in each cell in the matrix, we have three candidates: coming from left, top and top-left. In greedy search, the one generates the lowest cost will be chosen. Now with simulated annealing, the candidate choosing procedure becomes stochastic. The probability of choosing event e_i $(i \neq b)$ at time t is calculated by Equation 10:

$$P(e_i, T(t)) = \exp\left[\frac{-1}{T(t)}(L(e_i) - L(e_b))\right] \tag{10}$$

where e_b is the best candidate event with the lowest cost $L(e_b)$ among the three candidates. To choose from the three candidates, first pick a random probability $P_{rand}(0 \leq P_{rand} < 1)$, and filter out all $e_i(i \neq b)$ where $P(e_i, T(t)) < P_{rand}$. Then the next state is chosen randomly from the rest of the $e_i(i \neq b)$ and e_b. The equation above shows that the event e_i with a cost $L(e_i)$ closer to $L(e_b)$ are more likely to be chosen. When the $T(t)$ is close to 0, then the probability of choosing $e_i(i \neq b)$ is also close to 0. So, as the temperature decreases with time, it is more and more probable to choose the best event e_b. The cooling schedule is updated each time after all word pairs in the corpus get aligned. We keep looping through the corpus until the cost function converges or the temperature is almost 0. And there may be several rounds of greedy search after the simulated annealing if necessary.

3.7 Two-Part Code Model

The two-part code model is an improvement of the baseline model based on the fact that the number of changes in languages sharing the same ancestor should be small. Therefore we expect the global matrix to be *sparse* – only a small proportion of all event types E will actually occur in the final alignments.

The fundamental idea of two-part code is to code the whole data in a more clever way by the help of the code book. First encode which event types in E have non-zero counts, then only encode the part of E that have non-zero counts. Implied by its name, the code length of the two-part code model consists of two parts: the code length of the code book $L_{tpc}(CB)$ and the code length of the data given the code book $L_{tpc}(D|CB)$. So the total code length $L_{tpc}(D)$ is:

$$L_{tpc}(D) = L_{tpc}(CB) + L_{tpc}(D|CB) \tag{11}$$

and $L_{tpc}(CB)$ and $L_{tpc}(D|CB)$ can be calculated as follows:

$$L_{tpc}(CB) = \log(|E| + 1) + \log \binom{|E|}{|E^+|} \tag{12}$$

$$L_{tpc}(D|CB) = -\sum_{e' \in E^+} \log(c(e')!) + \log(\sum_{e' \in E^+} (c(e') + 1) - 1)! - \log(|E^+| - 1)! \tag{13}$$

For the code book, we first encode the number of event types with non-zero counts by $\log(|E| + 1)$ bits. Then we encode exactly which event types in E have non-zero

counts by $\log \binom{|E|}{|E^+|}$ bits, where E^+ is a subset of E that contains event types which have non-zero counts. After we have the code book, the second part is similar to the baseline model, the only difference here is that only the event types of non-zero counts are considered.

The probability of event ϵ is now different from the baseline model. We have to consider two cases: (1) the type of ϵ has already been observed before and (2) the type of ϵ has not been observed before. I omit the equation of how to calculate $P(\epsilon)$ here, but this will be discussed in detail after we introduce "kinds". See Equation 19 and Equation 21.

Practically, using two-part code model will generate better alignments as well as lower cost.

Further, we divide the global alignment matrix into three *kinds* as in Equation 14. Each kind represents one type of symbol alignment.

$$K = \{(symbol : symbol), (symbol : .), (. : symbol)\} \tag{14}$$

K is the set of event *kinds*. We can make the assumption that different *kinds* of alignment behave differently.

Now we can calculate the cost of code book and data given the code book for each kind separately, see Equation 15, 16.

$$L_{tpc-kinds}(CB) = \sum_{k \in K} \left[\log(N_k + 1) + \log \binom{N_k}{N_k^+} \right] \tag{15}$$

$$L_{tpc-kinds}(D|CB) = L(K) + \sum_{k \in K} \left[- \sum_{e' \in E_k^+} \log(c(e')!) + \log \left[\sum_{e' \in E_k^+} (c(e') + 1) - 1 \right] ! - \log(N_k^+ - 1)! \right] \tag{16}$$

where N_k is the number of event types in kind k, N_k^+ is the number of positive-count event types in kind k, $L(K)$ is the code length of encoding which *kind* each event instance belongs to.

Coding the four possible kinds prequentially, we get the cost of encoding the kind

of all events as:

$$L(K) = -\sum_{k' \in K} \log c(k')! + \log \left[\sum_{k' \in K} (c(k') + 1) - 1 \right]! - \log(|K| - 1)! \tag{17}$$

where the count of the kind k' is defined as $c(k') \equiv \sum_{e' \in k'} c(e')$.

To calculate the difference of the total cost after event instance ϵ has been observed, consider two cases:

Case 1:

The new event instance ϵ is of a type $\hat{e}$ that has already been observed before, and $\hat{e}$ is of kind $\hat{k}$. Let L_{new} and L_{old} be the cost after and before observing ϵ. Then the cost change can be calculated as:

$$\begin{aligned}
\Delta_\epsilon L &= L_{new} - L_{old} \\
&= L_{new}(CB) - L_{old}(CB) + \ L_{new}(\mathcal{E}|CB) - L_{old}(\mathcal{E}|CB) \\
&= L_{new}(K) - L_{old}(K) + \\
&+ \left[- \sum_{e' \in E^+_{\hat{k}}} \log(c(e')!) + \log[\sum_{e' \in E^+_{\hat{k}}} (c(e') + 1) - 1]! - \log(N^+_{\hat{k}} - 1)! \right]_{new} \\
&- \left[- \sum_{e' \in E^+_{\hat{k}}} \log(c(e')!) + \log[\sum_{e' \in E^+_{\hat{k}}} (c(e') + 1) - 1]! - \log(N^+_{\hat{k}} - 1)! \right]_{old} \\
&= -\log\left(c(\hat{k}) + 1\right) + \log \sum_{k' \in K} (c(k') + 1) - \log \frac{c(\hat{e})_{old} + 1}{\sum_{e' \in E^+_{\hat{k}}} c(e')_{old} + |E^+_{\hat{k}}|}
\end{aligned} \tag{18}$$

And the probability of ϵ can be derived as:

$$P(\epsilon) = 2^{-\Delta_\epsilon L} = \frac{c(\hat{k}) + 1}{\sum_{k' \in K} (c(k') + 1)} \cdot \frac{c(\hat{e}) + 1}{\left(\sum_{e' \in E^+_{\hat{k}}} c(\hat{e}) + |E^+_{\hat{k}}| \right)} \tag{19}$$

Case 2:

The new event instance ϵ is of a new type $\hat{e}$ that has not been observed before, and $\hat{e}$ is of kind $\hat{k}$

$$
\begin{aligned}
\Delta_\epsilon L &= L_{new} - L_{old} \\
&= L(\mathcal{E}|CB)_{new} - L(\mathcal{E}|CB)_{old} + L(CB)_{new} - L(CB)_{old} \\
&= L_{new}(K) - L_{old}(K) + \\
&+ \left[- \sum_{e' \in E^+_{\hat{k}}} \log(c(e')!) + \log[\sum_{e' \in E^+_{\hat{k}}} (c(e') + 1) - 1]! - \log(N^+_{\hat{k}} - 1)! \right]_{new} \\
&- \left[- \sum_{e' \in E^+_{\hat{k}}} \log(c(e')!) + \log[\sum_{e' \in E^+_{\hat{k}}} (c(e') + 1) - 1]! - \log(N^+_{\hat{k}} - 1)! \right]_{old} \\
&+ \left[\log \binom{N_{\hat{k}}}{N^+_{\hat{k}}} \right]_{new} - \left[\log \binom{N_{\hat{k}}}{N^+_{\hat{k}}} \right]_{old} \\
&= - \log \left(c(\hat{k}) + 1 \right) + \log \sum_{k' \in K} (c(k') + 1) \\
&+ \log \frac{\left[\sum_{e' \in E^+_{\hat{k}}} (c(e') + 1) + 1 \right] \left[\sum_{e' \in E^+_{\hat{k}}} (c(e') + 1) \right]}{c(\hat{e})_{old} + 1} + \log \frac{1}{N^+_{\hat{k}}} \\
&+ \log \frac{N_{\hat{k}} - N^+_{\hat{k}}}{N^+_{\hat{k}} + 1}
\end{aligned} \tag{20}
$$

Note: $c(\hat{e})_{old} = c(\hat{e}) = 0$

And the probability of ϵ is:

$$
p(\epsilon) = p(\hat{k}) \cdot \frac{N^+_{\hat{k}}}{\left[\sum_{e' \in E^+_{\hat{k}}} (c(e') + 1) + 1 \right] \left[\sum_{e' \in E^+_{\hat{k}}} (c(e') + 1) \right]} \cdot \frac{(N^+_{\hat{k}} + 1)}{(N_{\hat{k}} - N^+_{\hat{k}})} \tag{21}
$$

and $p(\hat{k})$ is calculated as:

$$
p(\hat{k}) = \frac{c(\hat{k}) + 1}{\sum_{k' \in K} (c(k') + 1)} \tag{22}
$$

As one might notice that both ALINE an Etymon project use dynamic programming to do word pair alignments. The main difference is, ALINE hand-picked the costs (or similarity scores) for different types of alignments, while in Etymon project the costs are learned during the aligning process, which is more fair and is not dependent on the choice of the values.

3.8 Context Model

The context model is a more complex model, and is also considered more intelligent. It takes the context, or environment into account in alignment modeling. More precisely, the cost of each alignment is affected by previous alignments. There are two main improvements in context model over the non-context ones: First, the alignments are done on the level of phonetic features instead of symbols. Second, the decision trees are used to encode the features in a way that reduce the entropy of the data most. This enables us to query previous alignments when aligning the current position.

3.9 Feature Data

In context model, the alignment of words are represented by the alignment of the phonetic feature vector of each symbol in the words. Figure 7 shows all the phonetic features and their values that are used in the context model. Note that the feature "TYPE" is not truly a phonetical feature, because it contains "Word Boundary" and "Empty Symbol", which are introduced for the coding procedure. The set of features used here follows the standard classification in phonological theory, the explanation of the features can be found in [ChH68]. Figure 8 illustrates the places of articulation of some consonants. For example, "bilabial" means the sound is made by touching upper and lower lips together, "dental" means the sound is articulated with the tongue against the upper teeth.

We encode the features in order. TYPE is the first feature to encode, because the following features are determined by it. There are 4 features for consonant symbols and 4 features for vowel symbols. For example the feature vector for the symbol "p" is **CPB-n**, it means that it is consonant, the manner of the sound is plosive, the place of articulation is bilabial and there is no secondary articulation.

3.10 Contexts

The context of feature f is defined as a tuple $(\mathcal{L}, \mathcal{P}, \mathcal{F}\,[\mathcal{X}])$, where $\mathcal{L}$ is the *level*: *source* or *target*. $\mathcal{P}$ is the position relative to f. Table 7 shows a list of possible positions. $\mathcal{F}$ is one of the features in Figure 7. $\mathcal{X}$ is the value of feature $\mathcal{F}$ and is optional. For example, here is one possible context (SOURCE, PREVIOUS SYMBOL, MANNER), which can be translated into a query: "what is the value of

Description	Feature Name	Values
Sound Type	TYPE	Consonant (C), Vowel (V), Empty Symbol (.), Word boundary (#)
Consonant Articulation	MANNER	Plosive (P), Nasal (N), Lateral (L), Trill (T), Fricative (F), Sibilant (S), Semi-vowel (W), Affricate (A)
	PLACE	Bilabial (b), Labiodental (l), Dental (d), Retroflex(r), Velar (v), Uvular (u)
	VOICED	Yes (+), No (-)
	SECONDARY	None (n), Palatalized ('), Labialized (w), Aspirated (h)
Vowel Articulation	VERTICAL	High (h), Mid-close (c), Mid-open (o), Low (l)
	HORIZONTAL	Front (F), Medium (M), Back (B)
	ROUNDED	Yes (u), No (n)
	LENGTH	Reduced (1), Very short (2), Short (3), Half-long (4), Long (5)

Figure 7: Phonetic features and values

the MANNER feature of PREVIOUS SYMBOL on the SOURCE level ?". This is called a general query. Because the answer is all values of $\mathcal{F}$, or all possible values of MANNER. Here is another example that includes $[\mathcal{X}]$: (TARGET, PREVIOUS VOWEL, HORIZONTAL, F), which means "Is the value of the feature HORIZONTAL of the PREVIOUS VOWEL on the TARGET level *Front* ?". This is called a binary query because the answer is *yes* or *no*. There are 2 levels, 7 positions, 9 features, so in total there are 126 general queries and 532 binary queries. Note that it is only allowed to query for the context that has already been encoded.

3.11 Building Decision Trees

All context information of the alignments is encoded in the form of a set of decision trees. We have one tree for each *feature* on each *level*. That means, 2 trees for feature TYPE, 8 trees for consonant features, 8 trees for vowel features, so in total 18 trees.

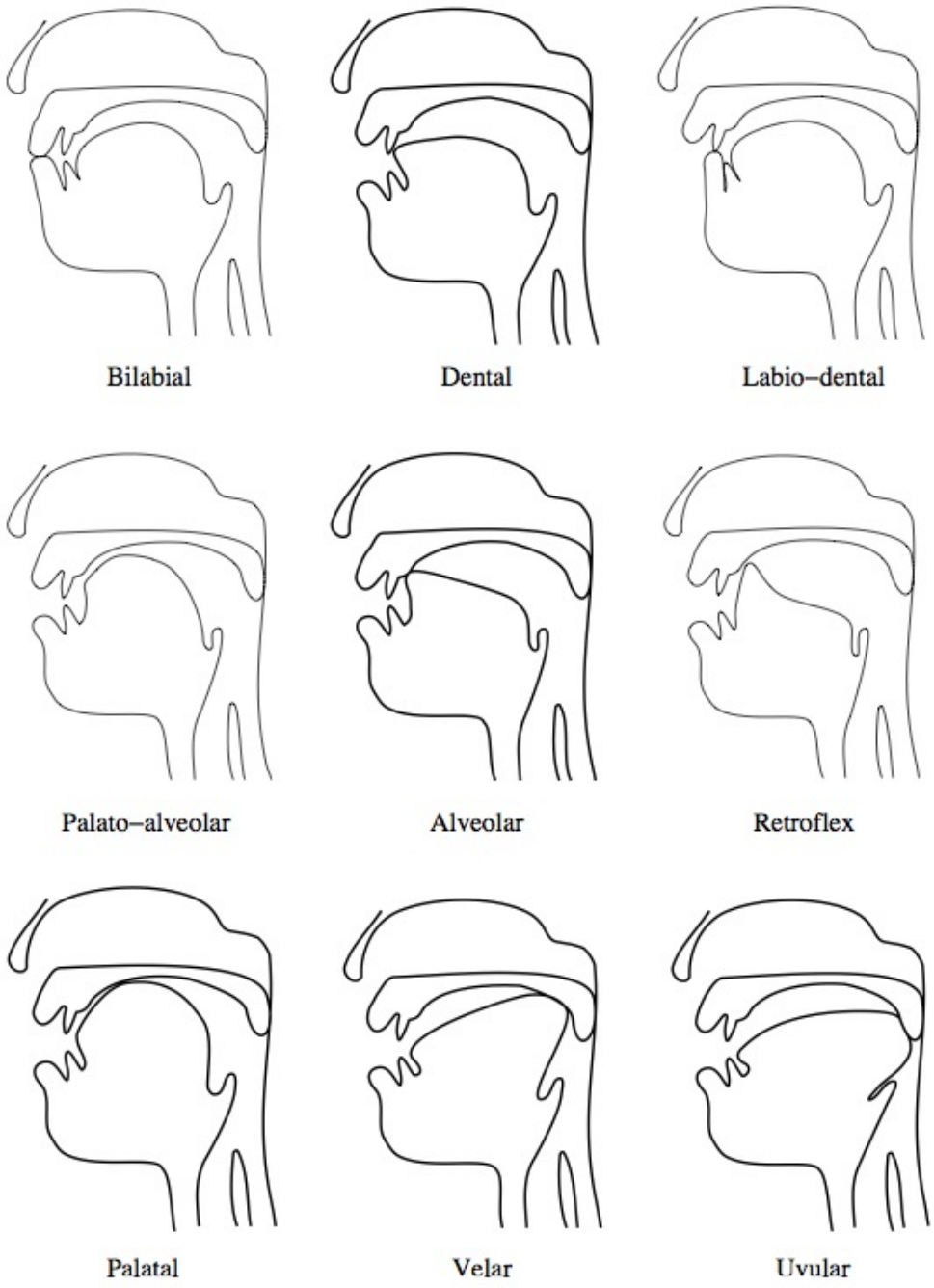

Figure 8: Places of articulation

Table 7: Possible positions for the context model

Position	Description
ITSELF	The current position, possibly dot
PREVIOUS SYMBOL	Previous non-dot symbol
PREVIOUS CONSONANT	Previous consonant
PREVIOUS VOWEL	Previous vowel
CLOSEST SYMBOL	Previous or self non-dot symbol
CLOSEST CONSONANT	Previous or self consonant
CLOSEST VOWEL	Previous or self vowel

First we initialize the trees by randomly align all word pairs, in the same way as baseline model. All trees are built in a certain order, starting at the SOURCE TYPE tree. Let's go through the procedure of building the TARGET HORIZONTAL tree. First, build the *root node*, which contains a *count matrix*. This matrix stores the counts of all the possible values of the feature HORIZONTAL, in this case *FRONT(F)*, *MEDIUM(M)* and *BACK(B)*, over all the symbols on TARGET level. Suppose the *root node* is:

Horizontal(target)	F	M	B
	530	50	490

Next, we try to find a way to split the *root node* using context to minimize the entropy of the count matrix. The context is chosen from all 658 contexts candidates (126 general queries and 532 binary queries). The *root node* is then split into branches by different values of feature $\mathcal{F}$. Each child node contains its own count matrix conditioned on $\mathcal{F}$. Here we compare the results of the *root node* split by two different context candidates: (SOURCE, ITSELF, HORIZONTAL) and (SOURCE, PREVIOUS SYMBOL, VERTICAL). The results are shown in Figure 9. As we can see from the two pictures, the split in (a) reduces the entropy more than the split in (b) does, thus will have a lower cost. The cost of the tree will be discussed later.

Note that "$\neq$" means the value does not apply. For example, if the symbol on the SOURCE level of ITSELF position is a consonant, then there is no HORIZONTAL or VERTICAL feature.

We keep splitting the trees until the tree cost, which is defined later in this chapter, stops decreasing. Each context candidate can only be used once in the splitting of a tree. And the best candidate is the one that reduces the tree cost the most.

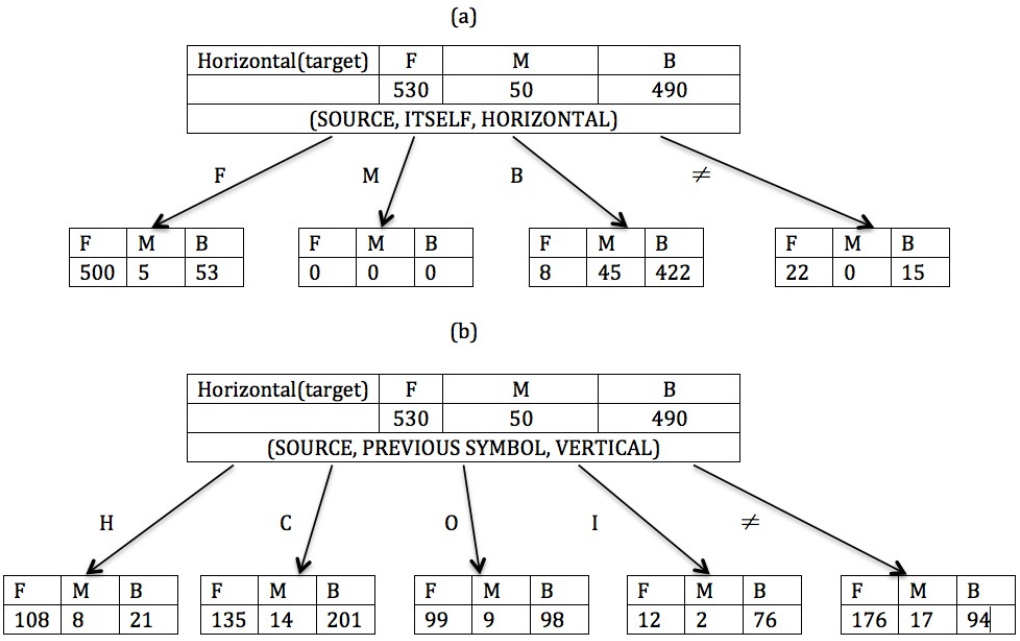

Figure 9: Decision tree splitting results for 2 different context candidates

The tree cost is calculated using Two-Part Code. The total cost of a tree X is defined as $L(X)$, which is the sum of the cost of the model $L_M(X)$ and the cost of the data given the model $L_{D|M}(X)$.

$$L(X) = L_M(X) + L_{D|M}(X) \tag{23}$$

The model cost $L_M(X)$ stores the tree structure, which is how the tree is built by making decisions. For each node n in the tree, the model cost of n is as follows.

$$L_M(n) = \begin{cases} 1 + \log(|Candidates|_{remain}) & when\ split \\ 1 & when\ not\ split \end{cases} \tag{24}$$

We use 1 bit to encode whether we split the node. If we split the node, we encode which candidate to choose from the remaining candidate set by $\log(|Candidates|_{remain})$ bits. We add up all the model cost for each node in the tree to get the model cost of the whole tree.

To calculate the cost of data given the model, we use prequential coding from Equation 3. $L_{D|M}(X)$ is then calculated by adding the costs of count matrices in all the

leaf nodes.

3.12 Context Model Alignment

The work flow of context model is very similar to the non-context ones. The steps can be summarised as follows.

STEP 1 Aligning all the word pairs randomly.

STEP 2 Build trees one by one in order based on the alignments. For example, the SOURCE-TYPE tree is the first one to build.

1. Build the *root node*.
2. Split recursively all the nodes.

STEP 3 Realignment.

1. Before doing realignment of each word pair, we remove the current contribution of this word pair from the trees by tracing down the trees according to the feature alignments and contexts. The structure of the trees are not changed, we only modify the count matrices in the *leaf nodes*.
2. Use dynamic programming to do the realigning. This is done in the same way as in baseline model. The only difference is that instead of using symbol-wise alignment to calculate the cost from the global matrix, we now use feature-wise alignment to calculate the cost from the trees.
3. Update new costs of re-aligned word-pair into the trees. Note here we only update the count matrices.
4. After all the word pairs get re-aligned , we rebuild the trees.

4 Experiments and Results

In this chapter, first I will describe some methods for model evaluation, then I will discuss about experiments I did related to the question marks in the data source.

4.1 Code Length and Data Compression

What the models are doing is actually compress the data with some coding scheme. So naturally the code length becomes one important criterion. According to information theory, the more regularity the data is learned, the more it can be compressed. Therefore we consider the model to be good if the code length is relatively small compared to other models. Most of the time we can see from the results of different models that the more "clever" ones get the lower code length and better alignments. Although the lower code length often goes with the better alignments, we found some exceptions in context models. Since our optimization function is code length, sometimes we get seemingly "non-sense" alignments with extremely good code length. The alignments in this case are often "shifted" in one direction, it seems that the model is so clever that it can find a special way to align the data, and use the context within those alignments to obtain a even smaller code length.The code length comparison of different models is shown below.

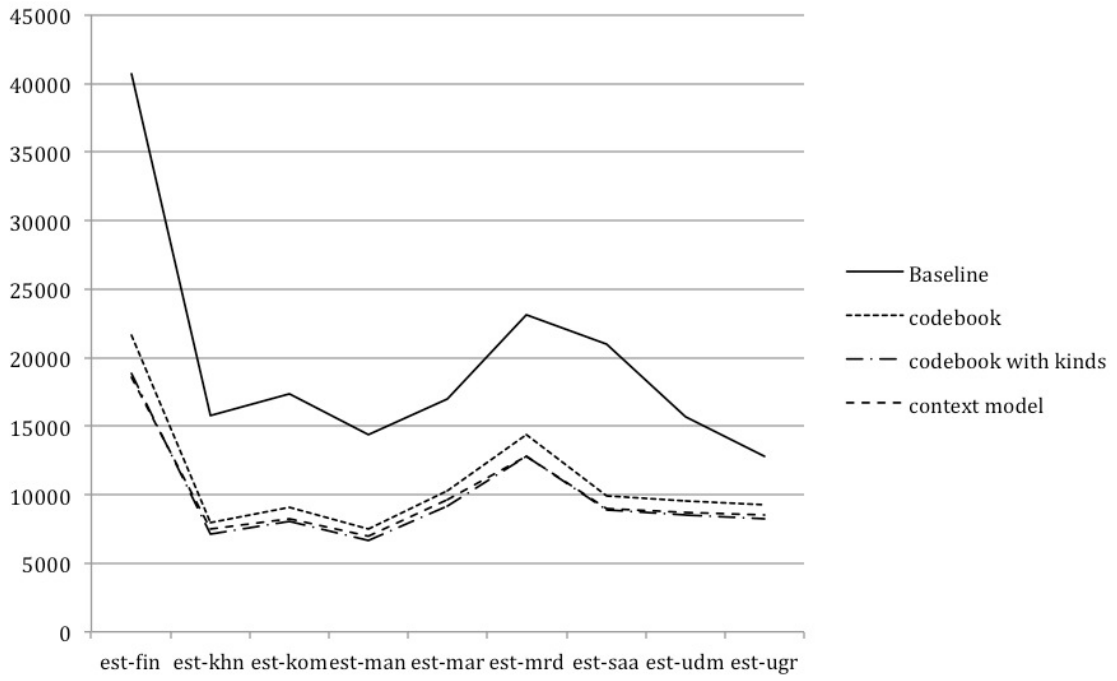

Figure 10: Code length of different models

Figure 10 shows code lengths for 4 models: Baseline Model, Two-Part Code Model,

Two-Part Code with Separate Kinds Model and Context Model. The code lengths are calculated from language pairs, which align Estonian to 9 other languages in StarLing database. We can see that the Baseline Model has the highest code length. The Two-Part Code Model, which introduces codebook, gains a great improvement. The Two-Part Code with Separate Kinds and Context Model are the best in terms of the code length.

Notice that here the code length is considered the raw cost, which is the code length of compressing all the data of a certain language pair. Raw costs are not comparable between different language pairs, because the number of words in them are very different. For instance, language pair with less word pairs will get a lower raw cost. In order to compare costs between different language pairs, we need to somehow *normalize* the raw costs.

4.2 Normalized Compression Distance(NCD)

In order to normalize the costs of different language pairs, we introduce the *normalized compression distance(NCD)*. We can use it to measure the "distance" between two languages. Based on [CiV05], the *normalized compression distance* of language a and b is defined as follows:

$$\delta(a,b) = \frac{C(a,b) - min(C(a,a), C(b,b))}{max(C(a,a), C(b,b))} \tag{25}$$

For most of the time $0 < \delta < 1$. $C(a,b)$ is the code length of compressing language a and b. Here we introduce the monolingual model, in which the language is aligned against itself. Note that in monolingual model, for example when aligning language a to itself, we are not using all the words in a to do the self-alignment, we exclude the words in a that do not have corresponding words in language b.

We then can create a *language distance matrix* by calculating NCD for each language pair. Table 8 shows what such matrix looks like.

We consider $C(a,b) \approx C(b,a)$ for non-context models, therefore the matrix can be treated as a symmetric matrix. But this is not the case for context models, $C(a,b)$ is asymmetric in context models because the order in coding two languages does matter.

We can then analyse the relations among different languages using the NCD matrix. We use the *neighbour joining*[SaN87] algorithm to build a phylogenetic tree of the

Table 8: NCD matrix for 10 Uralic languages (non-context model)

	est	fin	khn	kom	man	mar	mrd	saa	udm	ugr
est	0.0000	0.4603	0.8515	0.8799	0.8820	0.8211	0.7965	0.5727	0.8617	0.8717
fin	0.4603	0.0000	0.8739	0.8768	0.8887	0.8070	0.7506	0.4947	0.8648	0.8640
khn	0.8509	0.8735	0.0000	0.8113	0.6759	0.8526	0.7991	0.6920	0.8296	0.8385
kom	0.8806	0.8778	0.8114	0.0000	0.7929	0.8266	0.8210	0.7079	0.5460	0.8180
man	0.8820	0.8885	0.6761	0.7929	0.0000	0.8128	0.7865	0.6784	0.7819	0.8403
mar	0.8210	0.8069	0.8531	0.8264	0.8133	0.0000	0.7234	0.6284	0.8228	0.8140
mrd	0.7965	0.7503	0.7995	0.8212	0.7874	0.7235	0.0000	0.5371	0.7934	0.7747
saa	0.5735	0.4947	0.6931	0.7070	0.6784	0.6281	0.5374	0.0000	0.6547	0.7354
udm	0.8609	0.8647	0.8302	0.5461	0.7820	0.8225	0.7933	0.6548	0.0000	0.8256
ugr	0.8720	0.8637	0.8392	0.8180	0.8403	0.8143	0.7741	0.7370	0.8253	0.0000

languages. Phylogenetic trees are used in biology to show the evolutionary relationships among various biological species based upon similarities and differences in their physical and/or genetic characteristics. In our model, we treat different languages as different species. The phylogenetic tree building process can be summarized as follows:

Step 1 Start off with a star tree containing all the nodes. See Figure 11(a).

Step 2 Based on the original distance matrix, we calculate a Q-Matrix.

$$Q(i,j) = (r-2)d(i,j) - \sum_{k=1}^{r} d(i,k) - \sum_{k=1}^{r} d(j,k) \tag{26}$$

where $d(i,j)$ is the distance of node i and j in the original distance matrix, r is the number of nodes.

Step 3 Pick two nodes with the lowest value in the Q-Matrix and join them as *neighbours*. Then add a new node X and remove the two *neighbours* in the distance matrix after joining. For example, assume that $Q(7,8)$ is the smallest, then the tree will be split as in Figure 11(b).

Step 4 Calculate the distance between X and the rest of the nodes, fill in the distance matrix. Start the algorithm again from Step 1.

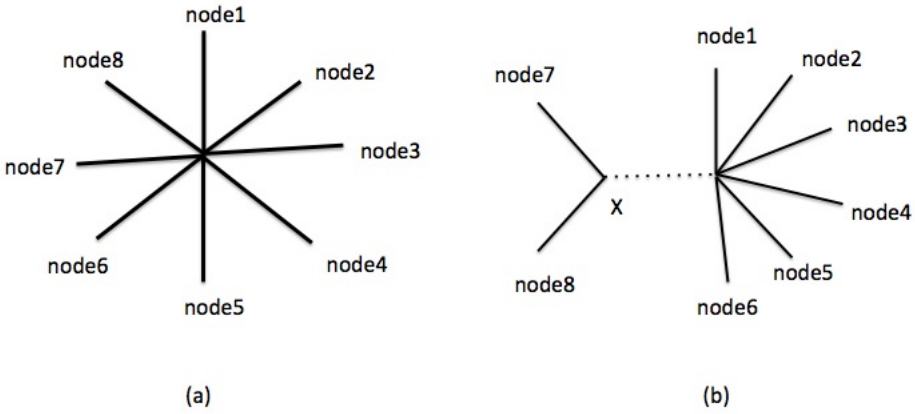

Figure 11: Neighbour Joining algorithm

Figure 12 shows the phylogenetic tree generated using neighbour joining algorithm and it is very similar to what linguists get.

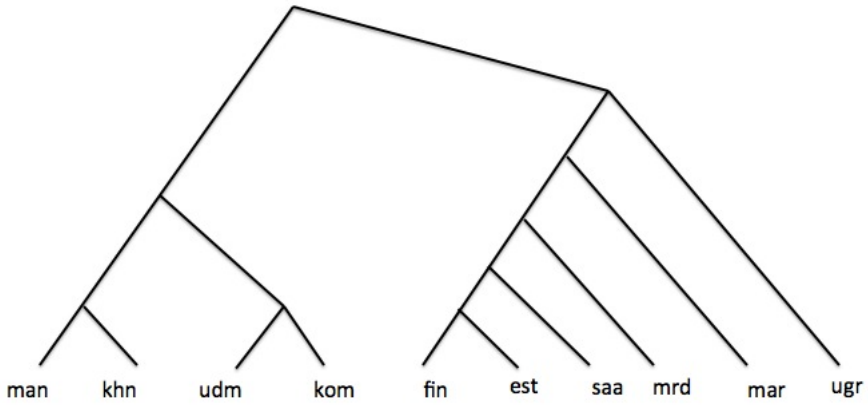

Figure 12: Phylogenetic tree of ten Uralic languages by neighbour joining.

4.3 Phylogenetic tree noise endurance experiment

The goal of this experiment is to test that whether it will improve the correctness of the tree if we average over several NCD matrices. Since there does not exist a truly correct "gold standard tree" to compare with in the real world, we came up with a way to simulate this process as follows:

1. Generate a NCD matrix using one of our models. Feed the matrix to neighbour joining algorithm to get a phylogenetic tree. We treat this tree as the *gold standard tree*.

2. Suppose the matrix contains n languages. Remove each language from the matrix to get n new matrices. By removing each language, just remove the row and column corresponding to that language. Now we get n new matrices each with $n-1$ languages in them.

3. Perturb the n new matrices by adding noise data to get n perturbed matrices. The noise data comes from normal distribution with standard variance σ and mean 0.

4. For each of the n perturbed matrices, generate a perturbed tree. Average the distances of each of the perturbed trees to the gold standard tree to get the $perturb-vs-gold-standard-distance$. For tree comparing method to calculate the distance of trees, we use TOPD/FMTS software [PGM07] with default parameters. TOPD/FMTS is software for comparing phylogenetic trees. It has been developed to calculate the differences between trees.

5. Make an averaged matrix from those n perturbed matrices, and generate an *average tree*.

6. Compare the *average tree* to the *gold standard tree* to get the *average-vs-gold-standard-distance*.

7. Get an array of $average-vs-gold-standard-distances$ and $perturb-vs-gold-standard-distances$ respectively, by changing the standard variance in the noise distribution.

8. Do the above steps 100 times and average the $average-vs-gold-standard-distances$ and $perturb-vs-gold-standard-distances$, plot them against different standard variances in the noise distribution, see Figure 13.

In Figure 13, the green line stands for the averaged *average-vs-gold-standard-distance*, and the red line stands for the averaged *perturb-vs-gold-standard-distance*. We can clearly see that the trees generated from the averaged perturbed matrices are more closer to the *gold standard tree* than the trees generated from individual perturbed matrices without averaging first. Figure 14 shows the result for 17 Uralic languages. As we can see, the result is better than the previous 10 languages in terms of noise

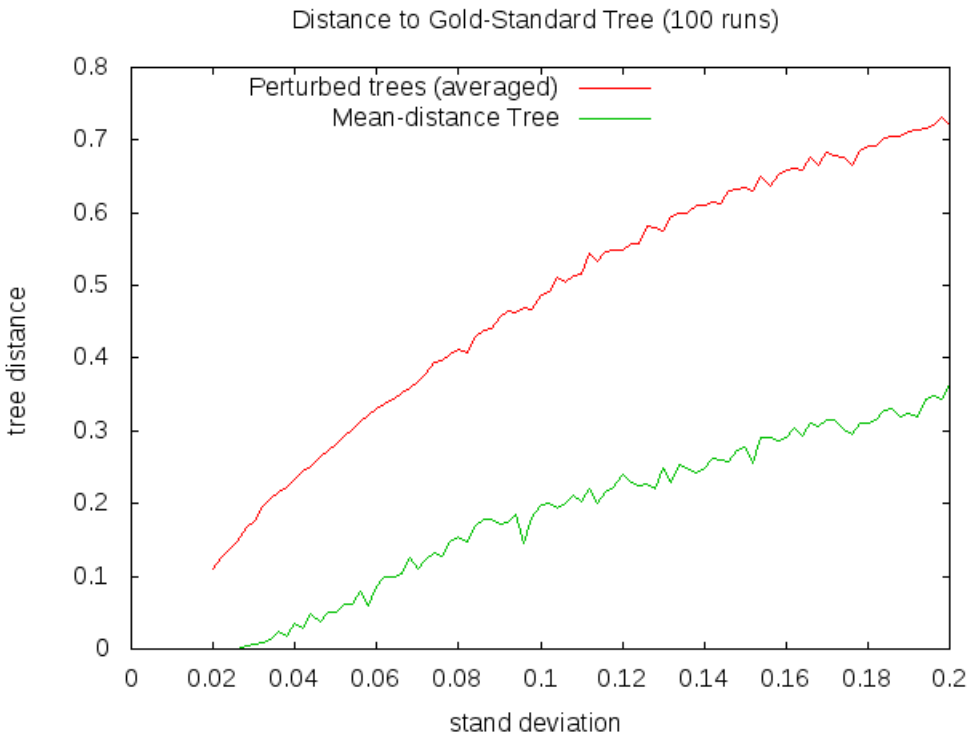

Figure 13: For 10 Uralic languages, plot average-vs-gold-standard-distance and perturb-vs-gold-standard-distance against the noise data standard variance $\sigma = [0, 0.2]$

endurance, when the standard variance of the noise is very high, the *average tree* is still very close to the *gold-standard tree*.

The result of this experiment shows that averaging the NCD matrices will help to get a more correct tree structure, and the more NCD matrices we do average on, the better tree we get.

4.4 Imputation

Imputation is another criterion for evaluating the performance of the models. The general idea is to *guess* or *impute* unseen data given the model. This is done by the following procedure. Given the data set of two languages L_1, L_2. Leave out one word pair (w_1, w_2) and build the model using the rest. Then given the word w_1, we are trying to *impute* the word w_2. We evaluate the goodness of our guess

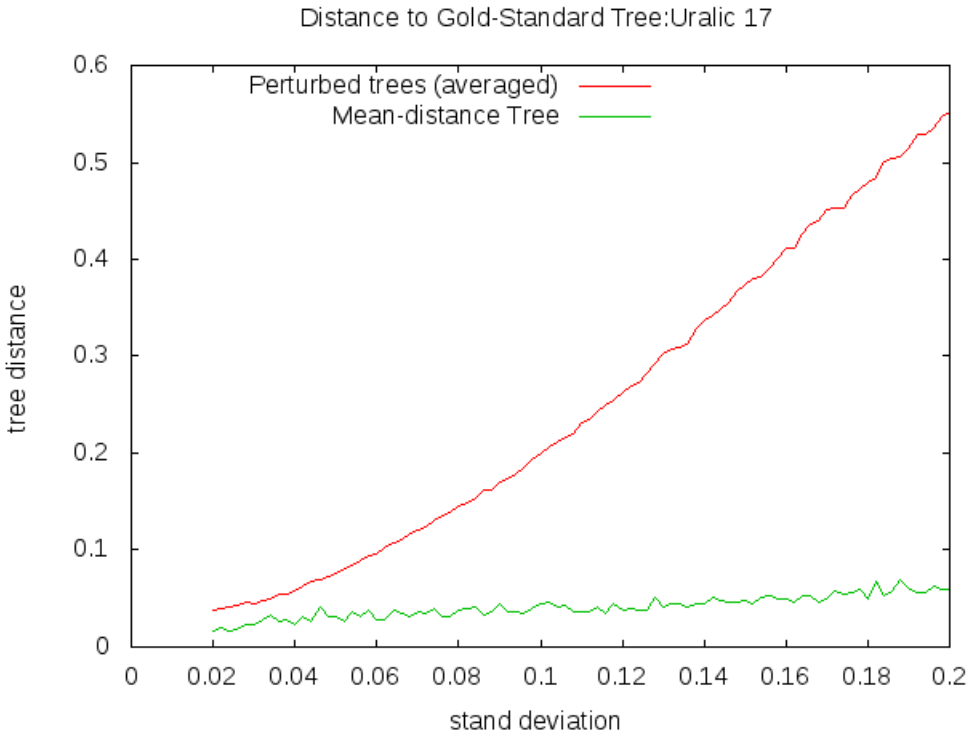

Figure 14: For 17 Uralic languages, plot average-vs-gold-standard-distance and perturb-vs-gold-standard-distance against a set of noise data standard variance $\sigma = [0, 0.2]$

by calculate the edit distance(e.g. Levenshtein distance) between the the original word and the imputed word. Repeat the same procedure for all word pairs. Finally, sum up the edit distances of all words and normalize them by the size of L_2. This is the Normalized Edit Distance(NED), by doing it feature-wise instead of symbol-wise we get Normalized Feature-wise Edit Distance(NFED). NED/DFED shows the predicting power of the model, it can be used not only to compare the goodness of different models, but also as a measure of language distance between different languages, as NCD.

The imputation procedure varies from model to model.

- **Non-Context Imputation** For non-context models, the imputation is easy. Since all the knowledge learned is stored in the global matrix M, we can do the imputation by simply search for most probable symbol alignments. This

is illustrated in Figure 15. For $s_i \in \mathbf{s}$, find the largest number in row i.

	.	t_1	t_2	...	t_j	...	$t_{\|T\|}$	#
.	-	$c(s_1, t_1)$	$c(s_1, t_2)$	...	$c(s_1, t_j)$	...	$c(s_1, t_{\|T\|})$	-
s_1	$c(s_1, -)$	$c(s_1, t_1)$	$c(s_1, t_2)$	...	$c(s_1, t_j)$	...	$c(s_1, t_{\|T\|})$	-
s_2	$c(s_2, .)$	$c(s_2, t_1)$	$c(s_2, t_2)$	...	$c(s_2, t_j)$	...	$c(s_2, t_{\|T\|})$	-
...	...	...	...	...	...	...	...	...
s_i	$c(s_i, .)$	$c(s_i, t_1)$	$c(s_i, t_2)$	...	$c(s_i, t_j)$		$c(s_i, t_{\|T\|})$	
...	...	...	...	...			...	-
$s_{\|S\|}$	$c(s_{\|S\|}, .)$	$c(s_{\|S\|}, t_1)$	$c(s_{\|S\|}, t_2)$	...	$c(s_{\|S\|}, t_j)$		$c(s_{\|S\|}, t_{\|T\|})$	-
#	-	-	-	-	-	...	-	N

Figure 15: Imputation of non-context models. To impute the target symbol given s_i: in line i, suppose the largest count is $c(s_i, t_j)$, then t_j is the imputed symbol on the target level.

- **Context model imputation** In context models we use the decision trees to do the imputation. The algorithm for context imputation is more complicated than non-context models. This can be found at [Yan13]. I will not discuss it here.

4.5 Leave-One-Out Score

Up until now, what we have been talking about is only how to make comparison among our models, and we need a way to compare our models to other probability models. But there exist several difficulties. First, we cannot compare the alignments because there is no gold standard that we can refer to. And comparing phylogenetic trees seems feasible but it is a weak comparison.

So here we introduce a new evaluation method, Leave-One-Out Score(LOO Score). LOO Score is a probability measurement, it measures the probability of our models' correctness. LOO Score is calculated as follows.

- **For each** word pair wp_i in data set:
  - Subtract the wp_i from the data set
  - Instead of impute the missing word, compute the probability assigned by the model to the correct one using Equation 8 and Equation 9.

- Sum up the scores for every word pair.

This procedure is similar to the calculation of the code length, and the code length and probability can be used interchangeably. With LOO Score, we now can compare our models to any probabilistic models.

4.6 Exploring the question marks in data source

There are question marks placed in some entries in the xml data source file. They mean some kind of uncertainty the linguists have about the word in that entry. Currently our program does not take them into account. But by exploring the question marks in the data, we probably can make a more "cleaned" version of the input data that leads to learning better models. And on the other hand, we also can test the level of uncertainty of the question marks given by the linguists, using our model.

In order for the computer program to utilize this piece of information, we need to find a way to represent the question mark information in the table-like input file. There are two steps:

- **Step 1** First we check each language field for a certain entry in the xml file, if it contains *global* question mark(s), which is the question mark(s) at the end of that entry. If so, the dialects of that language are not questioned among themselves. Co-index them with the same index number. For example:

```
<field name="SAA">vĺøńńe (oa) (T), vuəńń\He\h “Frau des älteren Bruders”</field>
<field name="MRD">niz-|ańa| “Schwiegermutter” (E), ańaka “ältere Schwester” (M) (?)</field>
<field name="UDM">ańį “Hanfgarbe” (J)</field>
```

  In the above picture, we can see that the language field "MRD" has 2 dialects: (E) and (M). And there is a question mark at the end of that line. So in this case, MRD(E) and MRD(M) will be given the same question mark index.

  If the question mark is only in one dialect, then only that dialect will be given a question mark.

- **Step 2** Go through all lines with more than one question marks in the table-like text file, check against the [Réd88] dictionary, if there are square brackets

grouping multiple languages and at the same time there is a question mark in front of the brackets, like:

... ?[LANG_A ... ; LANG_B ...] ...

and in the XML file LANG_A and LANG_B are both questioned. Then we will co-index the question marks for LANG_A and LANG_B, for instance, with **?a**. This means that the languages within the square brackets are not questioned among themselves.

This is only one interpretation of the question marks in the data, and more thoughts should be given in the future.

With the question marks tagged as described above, now we will test if the question marked cognates really have the meaning that we are expecting based on our models. The basic idea is that, in general, the questioned cognates should have looser connections comparing to the non-questioned ones. Questionable word pairs are word pairs with different question mark indexes, or one word with question mark and the other doesn't. The experiment is described in more detail as follows.

1. Train the model(non-context or context) on all the cognates in the data, including the questioned ones.
2. For each cognate pair do the following.
   (a) Remove the contribution of that cognate pair in the model(just as what we do in imputation).
   (b) Align that pair to get a cost from the Viterbi alignment.
   (c) Normalize this cost by NCD.
3. Rank all cognate pairs according to their NCD scores.
4. Check whether the questionable pairs end up toward the bottom of the ranking, which means that the two words in the pair are less related. We can see the results for aligning some language pairs in Figure 4.6.

We can see that for some language pairs we get convex curves, like KHN_ V and KOM_S, which means that the number of questioned cognates is increasing more rapidly when getting nearer to the bottom of the ranking list. This proves that our hypothesis might be right. But interestingly, the curves are

concave for some other language pairs, like KHN_V and MAR_KB. These are interesting results, which may indicate that for some languages, there exists special features of the data that we don't know. These could be useful results to present to linguists for further analysis.

5 Conclusion

In this thesis I discussed two topics related to cognate in computational linguistic: cognate alignment and cognates identification. For cognate identification, I did a brief survey and discussed several methods in the field. The methods mostly fall into two categories: orthographic and phonetic. The orthographic approaches are rather naive and straight-forward, and can be easily implemented. But the performance of phonetic approaches are often better. [Kon02] also include a lexical database called WordNet to assist the cognate identifying process. For cognates alignment I focused on the etymon project [Yan13]. The theory behind the project is to use MDL principle to optimize the aligning process of cognate sets. We imagine that we want to encode the whole aligned data and transmit it to others. According to information theory and MDL principle, the best alignment is one with the lowest code length. The alignments are done in different levels for different models. In baseline model and two-part code model, the alignments are done on the symbol level, while in context model the alignments are done on sound feature level. Several evaluation methods have been used to assess the performance of those models. The baseline model is the most basic model and has the highest cost, and the context model is the most complex model and with the lowest cost. Within the non-context models, the costs decrease from baseline to two-part code, but the actual cognate alignments are similar. I also discussed about utilizing question marks in the data source file. Though we can extract question marks from data source, more experiments and thought need to be done before it can be integrated into the existing models.

My contribution to the Etymon project mainly lies in the following parts. The development of the two-part code model with kinds (Chapter 3.7), the phylogenetic tree noise endurance experiments(Chapter 4.3), the implementation of leave-one-out score(Chapter 4.5) and exploring the question marks in data source(Chapter 3.6).

In all, cognate sets plays an important role in linguistic research and applications, and many novel methods are emerging. And I believe that more innovative methods will be found in the future.

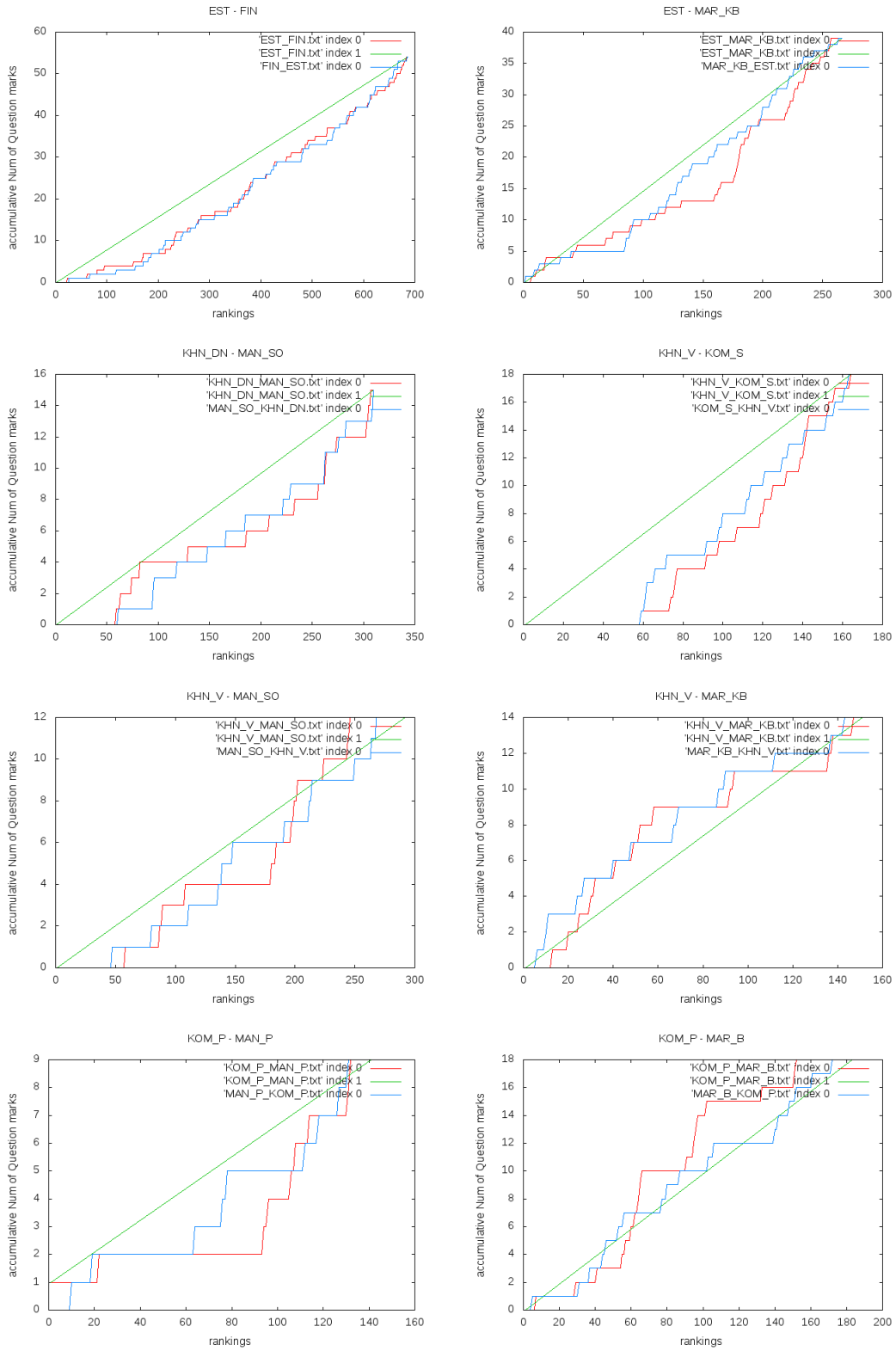

Figure 16: X-axis: rankings of the cognate pairs. Y-axis: accumulative number of question marks. Red-line and Blue-line represents different orders when aligning the language pairs. Context model is used here.

References

Bro90 Brown, P. F. et al., A statistical approach to machine translation. Technical Report, IBM, Thomas J. Watson Research Center, Yorktown Heights, NY, 1990.

ChH68 Chomsky, N. and Halle, M., *The Sound Pattern of English.* Harper & Row, New York, 1968.

CiV05 Cilibrasi, R. L. and Vitanyi, P. M., Clustering by compression. *IEEE Transactions on Information Theory*, 51,4(2005), pages 1523–1545.

Daw84 Dawid, A., Present position and potential developments: Some personal views: Statistical theory: The prequential approach. *Journal of the Royal Statistical Society. Series A (General), (1984)*, pages 278–292.

Fel98 Fellbaum, C., *WordNet: an electronic lexical database.* The MIT Press, 1998.

Grü07 Grünwald, P., *The Minimum Description Length Principle.* MIT Press, 2007.

ItK00 Itkonen, E. and Kulonen, U. M., *Suomen Sanojen Alkuperä (Origin of Finnish Words).* Suomalaisen Kirjallisuuden Seura, Helsinki, Finland, 2000.

Kes95 Kessler, B., Computational dialectology in Irish Gaelic. *In: Proceedings of the European ACL, 60-67. Dublin: Association for Computational Linguistics*, 1995, pages 60–67.

KGV83 Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P., Optimization by simulated annealing. *Science*, 220,1(1983), pages 671–680.

Kon02 Kondrak, G., *Algorithms for language reconstruction.* Ph.D. thesis, Toronto, Ont., Canada, Canada, 2002. AAINQ74607.

Lad95 Ladefoged, P., *A Course in Phonetics.* Cengage Learning, 2010.

LiV08 Li, M. and Vitányi, P. M. B., *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, 2008.

Mel99 Melamed, I. D., Bitext maps and alignment via pattern recognition. *Computational Linguistics*, 25, pages 107–130.

McO96 McEnery, T. and Oakes, M., Sentence and word alignment in the crater project. *In Thomas, J. and Shot, M., editors, Using Corpora for Language Research*, 1996, pages 211–231.

NeH97 Nerbonne, J. and Heeringa, W., Measuring dialect distance phonetically. *Proceedings of the Third Meeting of the ACL Special Interest Group in Computational Phonology*, 1997, pages 11–18.

Oak00 Oakes, M. P., Computer estimation of vocabulary in a protolanguage from word lists in four daughter languages. *Journal of Quantitative Linguistics*, 7,3(2000), pages 233–243.

PGM07 Puigbo, P., Garcia-Vallve, S. and McInerney, J. O., Topd/fmts: a new software to compare phylogenetic trees. *Bioinformatics*, 23,12(2007), pages 1556–1558.

Réd88 Rédei, K., *Uralisches etymologisches Wörterbuch.* Harrassowitz, Wiesbaden, 1988.

SFI92 Simard, M., Foster, G. F. and Isabelle, P., Using cognates to align sentences in bilingual corpora. *in Proceedings of the Fourth International Conference on Theoretical and Methodological Issues in Machine Translation*, 1992, pages 67–81.

Sha48 Shannon, C. E., A mathematical theory of communication. *The Bell System Technical Journal*, 27, pages 379–432, 623–656.

SaN87 Saitou, N. and Nei, M., The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4,4(1987), pages 406–425.

Sol64 Solomonoff, R., A formal theory of inductive inference, part l and part 2. *Information and Control*, 7,1(1964), pages 1–22, 224–254.

Sta13 Starostin, S. A., Tower of babel: Etymology databases. `http://newstar.rinet.ru/`. [5.7.2013]

Yan13 Yangarber, R. et al., Etymon project. `http://etymon.cs.helsinki.fi/`. [5.7.2013]