Date of acceptance Grade

Instructor

Minimum Description Length Modeling of Etymological Data

Suvi Hiltunen

Helsinki April 27, 2012

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
Suvi Hiltunen

Työn nimi — Arbetets titel — Title
Minimum Description Length Modeling of Etymological Data

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Master's Thesis	April 27, 2012	67 pages

Tiivistelmä — Referat — Abstract

Computational etymology develops automatic methods for the study of language change. Topics of interest are discovery of regular sound correspondences, determination of genetic relationships of languages and reconstruction of proto-languages. This work introduces probabilistic models for induction of alignment rules for etymological data. The models are applied to data from the Uralic language family.

The minimum description length (MDL) principle is a method for inductive inference and model selection. The underlying idea of MDL is that any regularity in the data can be used to compress it. We apply the MDL principle by encoding the etymological data using prequential coding and use the models to minimize the compressed size of the data. The result of the optimization is an alignment of the given set of genetically related words. Based on the alignment, we can study the rules of correspondence induced by the models. We also utilize it to compute the inter-language distances, which are used to reconstruct a phylogenetic language tree. The learning capability of the models is evaluated by letting them to predict unseen data. The results show that the models can describe the recurrent sound correspondences and measure the true relationships of the languages.

ACM Computing Classification System (CCS):
E. Data
    E.4 Coding and Information Theory
G. Mathematics of Computing
    G.3 Probability and statistics
I. Computing Methodologies
    I.2 Artificial Intelligence
        I.2.6 Learning
        I.2.7 Natural Language Processing
J. Computer Applications
    J.5 Arts and Humanities

Avainsanat — Nyckelord — Keywords
computational etymology, minimum description principle

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Computational etymology utilizes computational methods to study the relations of languages and origins of words. Typical research areas are identification of words that are from same origin, determination of genetic relations of languages and reconstruction of their ancestral forms. The distances of related languages can be quantified and used to model their evolutionary history. The guideline these methods use is the assumption of the *regularity of sound change*. This means that when the languages evolve, the phonemes of words undergo changes that depend on the structure of the words. Sometimes it concerns all the instances of the sound in a language, sometimes it can be environmentally conditioned, in which case it only occurs in a certain context. The discovery of these *regular sound correspondences* across languages is one fascinating task in which the computational method are useful.

If we are given a set of etymological data, how can we induct rules of regular sound correspondences? The problem can be seen as a task of finding an optimal *alignment* between the words from same origin. If we align two word forms from very closely related languages, there are probably very little replacements of sounds or gaps in the alignment. The longer time from the divergence of the languages, the harder it is to see the steps of language evolution in alignments without any additional information. Another way to say this: the closer the languages are to each other, the more consistent are the data sets and the more regularity there is in the alignments.

The *minimum description length* (MDL) principle is a method for inductive inference that is based on an idea that any regularity in the data can be used to compress the data [GMP05]. MDL gives a solution to the model selection problem, the best model is the one that can compress the most. If the data can be compressed, there exists some kind of a *description method* that can be used to uniquely describe the data in a short manner. The determination of the description method is a topic studied in *information theory*. There the objective is to find a code that is restrictive enough to be computable, but also general enough to be able to compress the regularity in the data.

The goal of this thesis is to present MDL-based models for alignment of etymological data. We apply several coding approaches and measure their ability to compress, since the compressed size of the data can be seen as the model cost. According to MDL, learning is compression, thus we try to find models that minimize the cost.

If a model is fixed, the model cost can be used to compare the quality of different data sets and to give a measure to group the relationships within a language family.

The key research question is to discover what kind of rules of correspondences it is possible to automatically discover from etymological data. We have developed models of different complexity to capture these rules: the models are used to find the best alignment at the symbol or sound level for etymologically related words of the given languages. The models are designed to be as general and objective as possible: the methods are developed to rely only on the data and to learn from data automatically in an unsupervised way. In the analysis, we use etymological data from Uralic language family, but the methods could be generalized to any group of related languages. The methods described in this thesis are based on the research done in the UraLink project that studies genetic relationships among the Uralic language family. My own contribution to the research is mainly on the implementation and design of the context-based models and their evaluation methods.

The structure of this thesis is as follows. In section 2 is presented prior work done in related application areas. The etymological data is introduced in section 3. The most important part of this thesis is in section 4, where the novel information theoretic models in computational etymology are defined and explained. In the next section 5 the evaluation methods used to compare the goodness of the models are described. Finally in 6 is discussion about the future work and conclusions.

2 Related work

All the new methods described in this thesis are based on the research done in the UraLink project during years 2010-2011. The intermediate results of this work are also published in several papers [WeY11, WHY11a, WHY11b]. This section describes the previous work done in the field of computational etymology (section 2.1). The following subsections cover the information theoretic methods that are utilized in our work. Finally, there are some notions about other application areas that utilize the minimum description length principle in section 2.5.

2.1 Computational Etymology

Before introducing the state of the art research in computational etymology, some important linguistic terms used throughout this thesis are first described. The exact definitions and more information can be found from [Ant89]. Most of the linguistic concepts described in this section belong to the subject of *comparative method*, which is a technique for comparing word forms that originate from a common language, in order to reconstruct the original ancestor language.

Cognates are words that have common etymological origin. They descend from the same ancestor language. The cognate words belong to the same language family and are said to be genetically related. *Cognate set* is a set of genetically related words in different languages of the language family. The work described in this thesis is based on modeling the cognate data.

The term *proto-language* is used either from a real, extincted language from which the other languages have descended or from a language which is artificially reconstructed by using the methods of computational etymology. In this work, the latter meaning is used, even though that is mainly a subject of future work. Based on the similarities and differences of corresponding sounds in the cognate sets, the *regular sound correspondences* between the related languages are determined. One of our goals in this work is to try to detect these regularities, by aligning the cognates. The study of *sound change* models the changes in the languages over time. Some of the sound changes are conditioned by the environment, which further complicates the task of language comparison.

The methods of *computational etymology* we are interested in are used to model the sound change and learning of regular sound correspondences. Some work is done

also in identification of cognates. That's something we are not aiming to, because our data sets consist of cognate sets. However, the task of finding cognates is very closely related to finding regular sound correspondences. That is the reason for the fact that the methods presented in this section often involve the search of cognates. The emphasis of the previous work in the field of computational etymology varies between the following three aspects:

- Finding cognates
- Finding optimal alignments of cognates
- Finding rules of correspondences

Since the goals above are so closely related, learning about one subject helps to understand the other aspects too.

The complexity of the problem varies depending on the restrictions set for the models. On one hand, if we are studying regular sound correspondences, the difficulty can be adjusted by determining how complex sound correspondences can be found. The easiest task is to look for one-to-one correspondences, while a more complex task is to extend them to many-to-many correspondences. On the other hand, the alignment can be done only between two languages, but it is more challenging to align multiple languages simultaneously.

Early work in the computational discovery of optimal alignments in linguistics is done by Michael Covington [Cov96]. He is using depth-first search algorithm and a cost function that is based on phonetic similarities to perform the alignment. The algorithm can be used only to find one-to-one correspondences and to align two languages at a time. The emphasis of the work is on finding the best alignments in phonetic sense. Later Covington has extended his method to handle more than two languages at once [Cov98].

Grzegorz Kondrak has done a series of papers. His early work [Kon02] is dealing with pairwise alignments. His purpose is to study regular sound correspondences and at the same time recognize cognates from data, since the problems are in a way very related. The first version of his CORDI algorithm works only with one-to-one sound correspondences. Later Kondrak extends his algorithm to recognize more complex sound correspondences [Kon03, Kon04]. Later papers by Kondrak are more survey-like [Kon09]. He has written also a review of different alignment methods that are based on phonetic similarities [KoS06]. Kondrak's work is based on

methods developed by Dan Melamed [Mel97, Mel00], who has worked on translation models. However, Kondrak has adopted Melamed's ideas in finding many-to-many sound correspondences and is using them in his algorithm.

Oakes [Oak00] has studied many aspects of comparative method, he aims at reconstruction of proto-languages. He is utilizing phonetic features, dynamic programming and determines regular sound correspondences in word lists.

Bouchard et al. [BLG07] do both alignments and reconstruction of evolutionary history of a language family in a probabilistic framework. This paper is closely related to our work, in various aspects. The probabilistic model they use is a Monte Carlo EM algorithm to fit the model parameters, which specify the the edits that govern how the words have evolved. A dynamic programming approach is used to reconstruct unknown word forms. Finally they evaluate the goodness of the model by computing the Levenshtein distance from the predicted reconstruction to the truth.

Bouchard et al. later evaluate their method for reconstruction of ancient word forms by means of imputation [BGK09]. Prokić et al. [PWN09] is aiming directly in aligning multiple languages at the same time.

One perspective to the study of the languages is the reconstruction of phylogenetic trees. We utilize this approach in the evaluation phase. Representative examples of work on this subject are e.g. [BWR09, NRW05, RWT02].

2.2 Information Theory

Information theory involves quantification of *information*. From the point of view of this work, the most important findings of the topic relate to the definition of fundamental limits of data compression and data communication.

Information theory was developed by Claude E. Shannon [Sha48]. Information can be understood to mean unexpectedness of events; the more surprise there is, the more there is information and the higher the probability of an event is, the lower is the information. More formally, information can be defined in the following way [Ham86]. Let us have a source alphabet S of symbols $s_1, s_2, \ldots, s_q$, each with probability $p(s_1) = p_1$, $p(s_2) = p_2$, ..., $p(s_q) = p_q$, $\sum p_i = 1$. Then the amount of information I can be defined as

$$I(s_i) = log_2 \frac{1}{p_i} \tag{1}$$

Entropy measures the expected value of the information contained in a message. A common measure unit of entropy is a bit. Entropy H is defined as:

$$H(S) = E(I(S)) = \sum_{i=1}^{q} p_i log_2 \left(\frac{1}{p_i}\right) \tag{2}$$

where E is the expected value, $I(S)$ information content of a discrete random variable S (the source alphabet) and p is the probability mass function of S. The entropy is maximal for an uniform probability distribution, and minimal for an extremely peaked distribution.

Information theory can also be viewed as a theory of communication over a channel [CoT91]. In this setting, a *sender* wants to send information using the channel to some *receiver*. The sender wants to encode the data in a way that the receiver is able to decode it, and also so that the sender has to send as few bits as possible. In this theoretical approach, the only thing the sender and the receiver have to share is the understanding of the coding method used and we need not care about noise and errors that exist in real word communication tasks.

The purpose of *source coding* is to compress source data as efficiently as possible so that fewer bits are needed in transferring the complete data, while keeping the the message completely decodable [CoT91]. Source codes try to reduce redundancies of the source and to squeeze more information to the bits that carry the information. A common principle of data compression algorithms is to minimize the average length of messages according to some particular probability model. The data is encoded in a way that the more frequent patterns in data have shorter code words than less frequent ones, cf. equation 2.

A prefix code is a code in which no code word is a prefix of any other code word [CoT91]. This means that all the code words are immediately recognizable and there is no need to code the end of a code word separately. *Kraft inequality* determines the relationship between code word lengths and existence of prefix codes, as defined in theorem 2.1. It tells when the lengths of the code words permit forming an prefix code, but it doesn't define the code itself. Kraft inequality allows the unification of codes and probability distributions if we define $p_x = 2^{-L(x)}$ for binary codes.

Theorem 2.1. *(Kraft Inequality) For any countably infinite set of code words x that form a prefix code, the code word lengths $L(x)$ satisfy the Kraft inequality*

$$\sum_{x} 2^{-L(x)} \leq 1 \tag{3}$$

Conversely, given any code lengths satisfying the Kraft inequality, we can construct a prefix code.

A prefix code is complete if there does not exist a shorter prefix code. A prefix code is complete if and only if the left side of the equation 3 is 1. The task of finding the prefix code with the minimum expected code length is equivalent to finding the set of lengths $L(x)$ that satisfy the Kraft inequality and whose expected length is less than for any other prefix code. The result of this optimization problem is that there exists a lower limit for the compression of the source code, and that is the entropy of the source code, $H(S)$.

2.3 Kolmogorov Complexity

Kolmogorov complexity is a theoretic measure of information of a data sequence, defined relative to a given universal Turing Machine. The Kolmogorov complexity of a sequence is the length of the shortest program that prints the sequence and then halts [Grü07]. A shortest program is a description method that maps each data sequence D to the shortest program that prints D and then halts. The lower the Kolmogorov complexity of a sequence, the more regular it is: Kolmogorov complexity is lower for more predictable sequences and higher for more complex ones. The theory and concept of Kolmogorov complexity was first discovered by Ray Solomonoff [Sol64], later also independently published by Andrey Kolmogorov [Kol65].

Kolmogorov complexity would be ideal to measure the informativeness of a given string by the size of the algorithm required to regenerate that string. In other words, it describes how much effort is required to generate the data [LiV94, Juo98].

There are two major problems why it is not possible to utilize the Kolmogorov complexity in practice. The first problem is that the Kolmogorov complexity cannot be computed. The second problem is that the description length of a sequence also involves a possibly large constant that depends on the description method used, i.e. the universal Turing Machine chosen [Grü07].

The connection between Shannon's entropy and Kolmogorov complexity is the fact that the former is an upper bound of the latter. [Juo98]. An intuitive approach to this non-trivial result is observation that a decompression program and a compressed file can be used to regenerate the original string.

There is also a connection between Kolmogorov complexity and the coding of etymological data. Kolmogorov complexity is uncomputable but it can be estimated for a given file using file compression programs, because a decompression program and a compressed file can be used to regenerate the original data sequence [LiV94]. This is also the intuition for our coding approach. We use code length to compare the goodness of our models and also the complexity differences between language pairs.

In general, the motivation to estimate the Kolmogorov complexity comes from the fact that if we find models that give us better compression of data,

- the better rules we find to describe the data,
- the better alignment we find for the data,
- the more tools we get to make the alignments more predictable,

and also the Kolmogorov complexity of the data gets lower. This idea was formulated in [KSL06].

2.4 The MDL Principle

Since Kolmogorov complexity is uncomputable, we need a formalization to approximate it. The answer is the *minimum description length principle* (MDL), introduced by Jorma Rissanen [Ris78]. MDL is based on the fact that any regularity in the data can be used to compress the data [Grü07]. In the MDL world, learning from data is data compression.

MDL is a method for inductive inference that gives a solution to the model selection problem. The model chosen by MDL is the one that leads to the best compression of the data. MDL is said to quantify Occam's razor. Where Occam's razor can only compare models of equal data fit and choose the simplest of those, MDL can compare any models of varying complexity and data fit and choose the one with the best trade-off between the two. Thus MDL automatically protects against overfitting. It can be used to estimate both the parameters and the structure of the model [Grü07].

The MDL approach gives a solution to the problems of Kolmogorov complexity. In principle, the idea is to scale down the task so that it becomes applicable [Grü07]. This is achieved by introducing a simple description method that is restrictive

enough to allow the computation of the length of the description, but is general enough to allow the compression of sufficiently many regular sequences. Naturally, the quality of an MDL method largely depends on the choice of this description method or model class. Any MDL model can be seen as an approximating Kolmogorov Complexity by replacing the universal Turing Machine with a model class suitable for the task at hand. This model class then is no longer universal, but instead it is efficiently computable.

Two-part code is the earliest and simplest implementation of MDL, and also the approach we adopt. In two-part coding, the best model $M \in \mathcal{M}$ is the one which minimizes the sum $L(M) + L(D|M)$, where

- $L(M)$ is the description length of the model, length in bits.
- $L(D|M)$ is the length of the description of the data that is encoded using the model, length in bits.

2.5 Applications of MDL Principle

In the field of natural language processing, the minimum description length (MDL) approach has been utilized in many research areas, e.g. part of speech tagging [RaK09], morphological analysis [WHY10, CrL02, Gol01], induction of grammars [AdJ06] and stemmatology [RHM06].

A particularly good example of applying the MDL principle, including prequential coding in a very similar manner as in this work but in a quite different application area, is our earlier work on induction of morphological structure of natural language [WHY10].

To the author's knowledge, MDL has not been utilized earlier in alignment of linguistic data. In other fields however this idea has been previously applied. For instance, in bioinformatics, sequence alignment has been done in information theoretic framework [Con07].

3 Etymological Data

Traditionally all etymological research is manually done by linguists. The results of the research are collected to handbooks of historical linguistics [Lyt73, Sin97], which contain rules and facts about the languages of interest and they provide a handful of examples of form x corresponds to y. However, these rules are not exact in a sense that they do not always specify all the exceptions, all precise conditions under which the rule holds and to what extend the rule is applicable.

As a summary, there are several problems in the historical databases that must be considered. First of all, they are lacking a quantitative measure of goodness for the rules. Because there has been a large amount of human interpretation involved in the creation of these databases, there is also unquantified amount of uncertainty involved. Typically different databases even conflict with each other.

3.1 Data

Our data is cognate data from Uralic language family, which has not been studied by computational means previously. The Uralic language tree is shown in figure 1. The tree illustrates the relations and distances of the languages in family. The more distant the languages are, the longer the time is they have evolved from the common ancestor language.

We have two Uralic etymological data sources in digital form. The first data resource is extracted from *SSA—Suomen Sanojen Alkuperä* ("The Origin of Finnish Words") [Itk00]. As the name implies, the dictionary is concentrated on the Finnish vocabulary and on the origins of Finnish words. SSA has more than 5000 Finnish entries, and for the related languages the number of entries varies mostly inversely proportionally to the distance of the language from Finnish. In the SSA, words are in so called dictionary form, meaning that they contain also morphological affixes, which are not relevant to reconstruction of genetic relations of languages.

The second and more important data resource for this project is *StarLing Database of Uralic Languages* by Russian Academy of Sciences [Sta05], which is originally based on [Réd91]. Table 1 gives an insight to the actual StarLing data, there is shown two cognate sets to demonstrate the properties of the cognate sets. StarLing differs from SSA in several respects. The StarLing data is more suitable for reconstruction purposes because it gives a stemmed or lemmatized form for most of the cognate

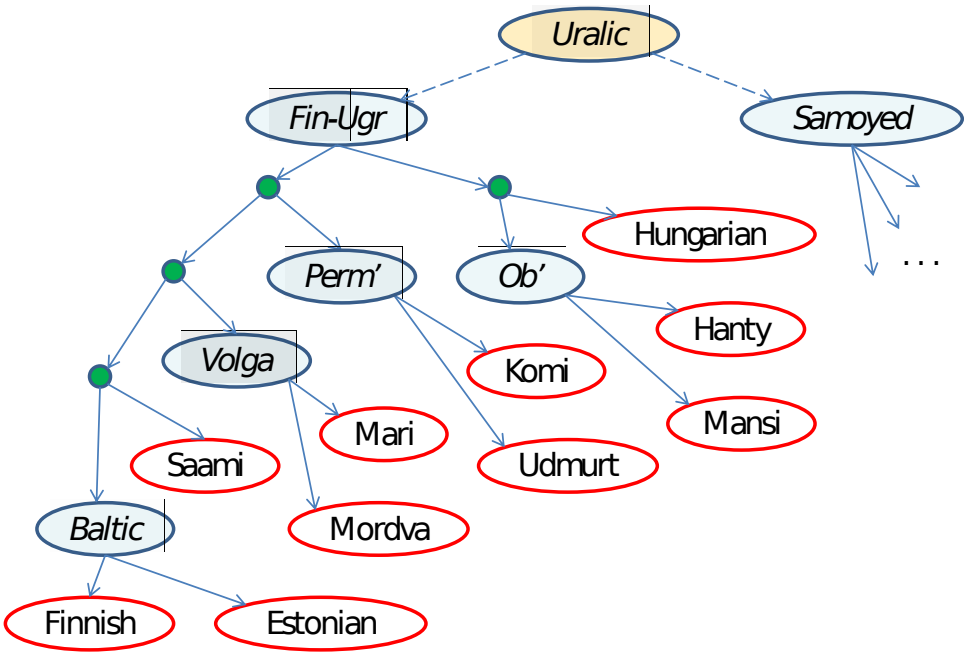

Figure 1: The Finno-Ugric branch of Uralic language family, according to [Ant89].

sets and it tries to normalize the words to the common base form. The data is more evenly distributed than SSA and it also contains cognate sets that do not have a Finnish member. The number of cognate sets in StarLing is about 2000. In figure 2 is illustrated how the amount of data is distributed among languages in both StarLing and SSA. The StarLing data set can be further preprocessed and the main dialects of each language (if any) can be extracted. This step improves the quality of the data but the flip side is the diminishing sizes of the cognate sets.

The language data in both databases is written using *Finno-Ugric transcription system* which is a phonetic transcription used for the transcription and reconstruction of Uralic languages. *Transcription* in linguistics is the systematic representation of spoken language in written form. It is a tool to consistently represent the sounds with written symbols. However, some of the languages (e.g. Hungarian and Saamic languages) use their own transcription systems that differ from the system used by other languages.

Language	**Cognate set 1**	**Cognate set 2**
Estonian	kolm	ala
Finnish	kolme	ala
Hungarian	három	al-
Khanty	kɔləm **V**, χutəm **DN**, χoləm **O**	i̯l **V**, it **DN**, il **O**
Komi	kujim **S P**, kvim **PO**	-ul **S**, -i̯v **P**, ul **PO**
Mansi	kōrəm **TJ**, χūrəm **So.**, kūrəm **P**	jalē̯k **TJ**, jalχ **KU** jalk **P**, jolik **So.**
Mari	kə̑m **KB**, kum **U B**	ül- **KB B**, ülö **U**
Mordvin	kolmo **E**, kolma **M**	al **E M**
Saam	gǫlbmâ **N**, kål'må **L**, kolm **T Kld. A**, koum **Not.**	vuolle **N**, -vuollē **L**, v¯ĭlle **T**, vūln **Kld.**, vueln **Not.**
Udmurt	kwiń **S**, kŭń **K**, kwiń **G**	ul **S**, ulə̯ **K**, ul **G**
English meaning	three	space below smth., below

Table 1: Two unrelated cognate sets from StarLing database. Each language entry contains a form for each dialect of the language, if known. In cognate set 2, the database gives several word forms with slightly different meaning for most of the words, here is shown only the first cognate mentioned, since the latter ones are ignored in our experiments. The dialects are marked with boldface, e.g. Mordvin language has two main dialects, **E** and **M**: E denotes Erzya dialect and M Moksha dialect.

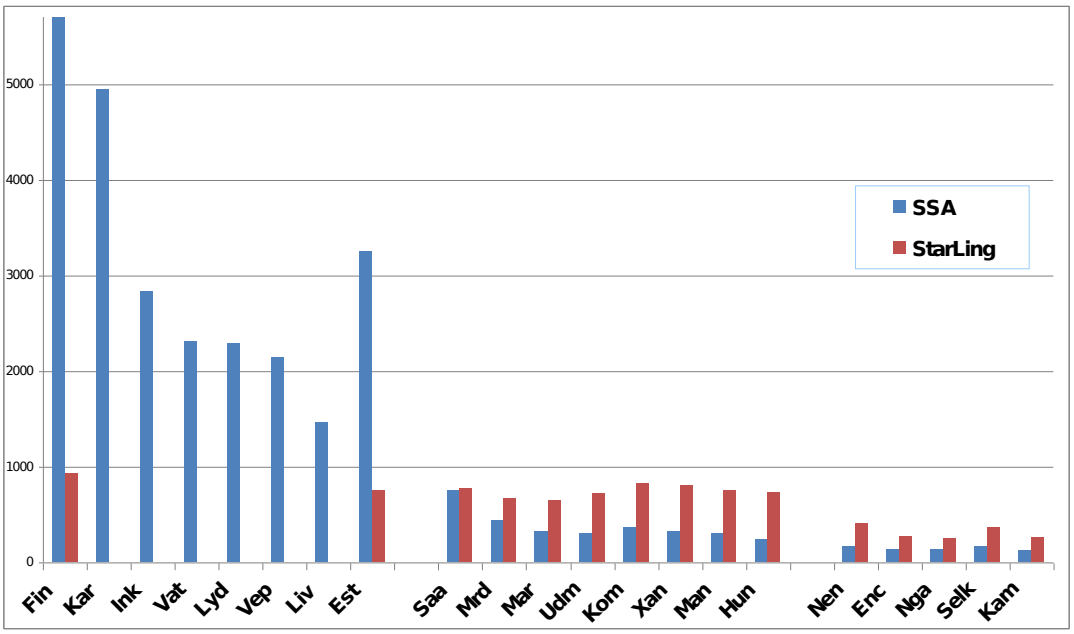

Figure 2: The distribution of data in SSA and StarLing. SSA is very Finnish-centric, whereas StarLing data is more evenly distributed.

3.2 Feature Data

We can get more information out from the data by further analyzing the phonological structure of the sounds of the languages. The phonological structure of the sounds can be expressed by converting the phonemes to vectors of phonetic features. The set of features we use follows the standard classification in phonological theory, for more information see e.g. [ChH68]. Table 2 gives examples of mappings to feature space for some phonemes. The table on left shows how the character modifiers are designed to reflect the phonetic variation. For example, if we simply compare glyphs **e** and **ē**, we can only say that they do not represent the same glyph. In feature space, the only difference between the glyphs is the length of the vowel. The table on right gives feature representations for some common consonants in the language family.

	e	**ē**	**e̯**	**ə**
TYPE	V	V	V	V
VERTICAL	c	c	c	c
HORIZONTAL	F	F	M	F
ROUNDING	n	n	n	n
LENGTH	3	5	5	1

	k	**p**	**r**
TYPE	C	C	C
MANNER	P	P	T
PLACE	v	b	d
VOICED	-	-	+
SECONDARY	n	n	n
LENGTH	1	1	1

Table 2: Feature vectors for some sounds. Table 3 gives the meaning for the feature values. E.g. sound **e** has a feature vector `VcFn3` that states that it is a vowel, it is mid-close and front and not rounded and it is mid-length.

All features and their possible values are shown in table 3. As summarized in the table, there are 4 features that can be used to describe any vowel, and 5 features for any consonant. The `TYPE` feature is special because it is not a phonetic feature. It is needed since we are not coding the words but the alignments, and we need a way to represent word boundaries and alignments with empty symbols, as explained in section 4. The features are always in the same order, which is the order shown in table 3. `TYPE` feature is in the first position, and it determines the features in the remaining positions.

As a result, in each language, all symbols of the corresponding alphabet will have an unique feature vector representation. Because of practical implementation related reasons, the conversion to feature space is done so that there must be a bijective function between each sound and the corresponding feature vector representation. Because of this requirement, the alphabet of a language has to be simplified to meet this condition in some cases, due to the too limited feature space. Mostly, the overlapping sounds occur only once or twice in the data and often seem to be inconsistencies in the data.

	Feature name	**Values**
Sound type	TYPE	Consonant (C), Vowel (V), Dot (·), 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)
	LENGTH	Short (1), Long(2)
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)

Table 3: Phonetic features and their values.

4 MDL-based Models for Alignment of Etymological Data

The goal of this work is to understand what kind of etymological phenomena may be discovered automatically from raw data. This involves finding the globally optimal alignment for the complete data: the task is to discover the complete alignment system with rules of regular sound correspondence. The concept of alignment is formally explained in section 4.1.

According to the Minimum Description Length principle which is defined in chapter 2.4, the best alignment is the one that can be encoded using as few bits as possible. The length of the code is our objective function and it is the way to define how to measure the quality of any given alignments. We use *prequential coding* to form the objective. More about prequential coding in section 4.2. Different models have different objectives, but they are all based on prequential coding.

The next step is to define the models we use. The baseline model is derived using some initial simplifications. First simplification is that we do only pairwise alignments for two languages at a time. Second simplification in the baseline model is the *1x1 alignment* where one source symbol may correspond only to one target symbol. Baseline model and the outline of the algorithm is explained in section 4.3.

The baseline model is followed by more complex models where these constraints are relaxed. Instead of pairwise alignments, it is possible to do a multi-dimensional alignment, where more than two languages are aligned at a time. Another way to extend the baseline model is align two languages at a time, but perform a $2x2$ alignment, where up to two symbols are aligned at the same time. The modifications to the baseline model are described in sections 4.5 and 4.6. The third extension to the baseline model is the *context* which defines how the environment conditions the alignments. More about context model is in section 4.7.

4.1 Aligning Pairs of Words

In a pairwise alignment a set of pairs of words from two related languages are aligned. The simplest form of an *alignment* means determining for each pair of words which symbols correspond to each other. We call the first language of the alignment pair *source* language and the second one *target* language. More formally, we denote an alignment at the symbol level with a pair $(s : t) \in S \times T$ where s is a single symbol

from the *source alphabet* S and t is a symbol of *target alphabet* T. Respectively, the sizes of the alphabets are $|S|$ and $|T|$, and the alphabets consist of the sounds of the corresponding languages.

We can further define a *source word* $\mathbf{s}$ and a *target word* $\mathbf{t}$. Both source and target words are then vectors of symbols of the corresponding alphabet. When we allow only pairwise symbol alignments, we restrict the source word and target word to the equal length where $|\mathbf{s}| = |\mathbf{t}|$. This is obviously too restrictive definition. Thus, we also need to allow *insertions* and *deletions*. For this purpose, we add and empty symbol, denoted by a dot, to both alphabets. The extended alphabets are called $S_{\cdot}$ and $T_{\cdot}$.

4.1.1 1x1 Symbol Alignment

Using the extended alphabets, it is now possible to align pairs of words of different length. An example case illustrated here is between the cognates *kaikki* - *kõik*, meaning "all" in Finnish and Estonian. The following alignments are both possible and without any further information, they seem both correct. The alignment on the left below consists of the pairs of symbols: (*k:k*), (*a:õ*), (*i:i*), (*k:k*), (*k:.*), (*i:.*).

k	a	i	k	k	i		k	a	.	i	k	k	i
|	|	|	|	|	|		|	|	|	|	|	|	|
k	$\tilde{o}$	i	k	.	.		k	.	$\tilde{o}$	i	k	.	.

Since there always exist many possible alignments for a word pair, we need a strategy for determining the quality of the alignments. In the example above it means that we need a consistent way to tell which of the two alignments shown is better.

Generally speaking, the best alignment refers to the *globally optimal* alignment. This involves finding as many as possible regular sound correspondences among all the aligned pairs in the data.

The complete alignment can be described with the help of *global count matrix* G, which stores the number of times each symbol pair was aligned in the complete corpus. In figure 3 is illustrated an abstract level count matrix. The rows of the matrix correspond to the source symbols from source alphabet, columns to the target symbols. We also introduce a new symbol # to denote the word boundaries. In each cell of the matrix is the count c of each symbol pair in data, that is $G(s_i, t_j) = c(s_i : t_j)$. The impossible alignments are marked with symbol -. Special alignment (.:.) is

	.	t_1	t_2	...	t_k	...	$t_{\|T\|}$	#
.	-				$c(. : t_k)$			-
s_1	$c(s_1 : .)$	$c(s_1 : t_1)$						-
s_2								-
...								-
s_k			$c(s_k : t_2)$		$c(s_k : t_k)$			-
...								-
$s_{\|S\|}$							$c(s_{\|S\|} : t_{\|T\|})$	-
#	-	-	-	-	-	-	-	n

Figure 3: Global count matrix stores in each cell the count of corresponding symbol pair alignments.

never allowed and since we use the word boundary symbol only at the end of each word, and a word boundary symbol can only be aligned with another word boundary symbol, the count of $(\# : \#)$ alignments is n, the number of word pairs in data.

4.1.2 2x2 Symbol Alignment

We can extend the 1x1 symbol alignment to allow alignment of maximum two symbols on both source and target side. For this purpose, we have to introduce two new symbols, ^ to denote a beginning of a word and \$ to symbolize the end of a word. If single source symbols $s, s' \in S_.$ and target symbols $t, t' \in T_.$, the set of possible symbol alignments becomes the following:

$$
\begin{array}{llll}
(\hat{} : \hat{}), & (\hat{} : \hat{}t), & (\hat{}s : \hat{}), & (\hat{}s : \hat{}t), \\
(. : t), & (s : .), & (ss' : .), & (. : tt'), \\
(s : t), & (ss' : t), & (s : tt'), & (ss' : tt'), \\
(\$: \$), & (\$: t\$), & (s\$: \$), & (s\$: t\$)
\end{array}
$$

The symbols that denote the end and the beginning of a word, behave differently from the phonetic symbols, since they can only be aligned with themselves or with a symbol pair containing them. An alignment between Finnish-Estonian word pair *nuoli-nool* (English meaning is "arrow") represents the 2x2 symbol alignment in practice. The example shows us some typical properties of the two languages: symbol pair $(\hat{}n, \hat{}n)$ shows that word-initial n occurs with another word initial n in both languages, (uo, oo) that diphthongs may align with long vowels and $(i\$, \$)$ that word final vowel in Finnish may disappear.

$$\begin{array}{cccc} \hat{}n & uo & l & i\$ \\ | & | & | & | \\ \hat{}n & oo & l & \$ \end{array}$$

Another example word pair *odotta-odota*, "to wait" in English", shows how a geminate consonant *tt* in Finnish becomes a single consonant *t* in Estonian:

$$\begin{array}{ccccc} \hat{}o & d & o & tt & a\$ \\ | & | & | & | & | \\ \hat{}o & d & o & t & a\$ \end{array}$$

4.1.3 3-Dimensional Alignment

In principle, the 1-to-1 model can be extended to any number of languages. The previously introduce models align data in 2-dimensions. Here we involve the third dimension for the 1-to-1 model. A 3-dimensional alignment is a triplet $(s : t : v) \in S_. \times T_. \times V_.$, where S, T and V are alphabets of the language family. An example of 3-dimensional alignment between Finnish-Estonian-Mordvin cognate triplet *yhdeksän-üheksa-veχksa* (meaning "nine" in English) is the following:

$$\begin{array}{ccccccccc} y & . & h & d & e & k & s & ä & n \\ | & | & | & | & | & | & | & | & | \\ ü & . & h & . & e & k & s & a & . \\ | & | & | & | & | & | & | & | & | \\ v & e & \chi & . & . & k & s & a & . \end{array}$$

Like the previous example shows, we are looking for 1-to-1 sound correspondences. Again the special alignment $(. : . : .)$ is not allowed, but it is possible to align a dot in two of the languages simultaneously. In 3-dimensional alignment, we accept as input data all the cognate sets where words from at least two languages are present. In this way, the previous example was *complete*: all 3 word forms are present whereas the following example is *incomplete*, since the word from the second language is missing. The languages in the next example are again Finnish, Estonian and Mordvin, the

meaning is “ghost”:

$$\begin{array}{ccccc} h & a & a & m & u \\ | & | & | & | & | \\ - & - & - & - & - \\ | & | & | & | & | \\ \check{c} & . & a & m & a \end{array}$$

We use *marginal* approach to build the global count matrix for the complete alignment of the data. This means that we need three marginal 2-dimensional matrices to store the all the pairwise alignments. These count matrices are called G_{ST}, G_{SV} and G_{TV}. In case of an incomplete example, we need only one marginal matrix to save an alignment count, exactly like in 2D-case, e.g. $c(s : - : v) = G_{SV}(s, v)$. In case of a complete example, all 3 marginal matrices need to be updated. When defined like this, the count of a complete example becomes $c(s : t : v) =$
$G_{ST}(s : t) + G_{SV}(s : v) + G_{TV}(t : v)$. Since each symbol is now coded twice, the counts are not comparable to the two-dimensional models.

4.2 Prequential Coding

This far we have defined our data corpus of n word pairs: $D = (\mathbf{s}_1, \mathbf{t}_1), \ldots, (\mathbf{s}_n, \mathbf{t}_n)$. We follow the Minimum Description Length principle (MDL) and want to *encode* all aligned symbol pairs as compactly as possible to find the globally optimal alignments. We use *prequential coding* [Daw84] to code the complete data. The prequential interpretation of MDL selects the model that has the best predictive performance when sequentially predicting unseen data [GMP05]. The same coding scheme can also be derived using Bayesian marginal likelihood [KMT96].

We start by choosing an alignment of each word pair, $(\mathbf{s}_i, \mathbf{t}_i)$, where $i = 1, \ldots, n$. We use these word pairs to *transmit* the sequence of pairwise symbol alignments, alignment *events*, from a sender to a receiver. An event ϵ is a pair of symbols $(s : t)$, in the event space $\mathcal{E}$:

$$\mathcal{E} = S_\cdot \times T_\cdot \cup \{(\# : \#)\} \setminus \{(. : .)\} \tag{4}$$

The use of word boundary symbol # can now be explained; we need it in order for the code to be uniquely decodable since it separates the words from each other.

The coding process can be illustrated with a prequential “coding game”. In the coding game, there are two participants, sender and receiver. The sender has a

mission to communicate a set of data to the receiver. In this game, both sides know the data structure beforehand, in our case, the alphabets. The data itself, however, needs to be communicated explicitly.

Let's say there are N different event types $e_i \in E$, with corresponding counts $[c_1, c_2, ..., c_N]$, where $E \subseteq \mathcal{E}$ is the set of possible event types. In initial, simplified form of this game, uniform (Dirichlet) priors are used. This means that prior count $\alpha_i = 1$ is given for each event type e_i, where $i = 1, \ldots, N$.

Both sender and receiver keep track of the number of different events sent and received by updating the counts of different event types and the total count of all events. This way we actually update a *probability distribution* as illustrated in table 4. Initially, the probabilities of all event types are uniformly distributed, $P(e_i) = 1/N$ for all event types e_i, like shown on the first row of the table. When an event is sent, the counts and the probability distribution is updated. The table 4 shows how the probabilities of event types e_1, e_2 and e_N change when different amounts of data is sent. The last row shows the probabilities after all the events are sent.

Finally, when all events are communicated, it is possible to compute the total cost of sending all the events. The probability of sent events can be computed by multi-

	$\mathbf{P(e_1)}$	$\mathbf{P(e_2)}$	$\mathbf{P(e_N)}$
Initial state	$\frac{1}{N}$	$\frac{1}{N}$	$\frac{1}{N}$
After the first event of type e_1	$\frac{2}{N+1}$	$\frac{1}{N+1}$	$\frac{1}{N+1}$
After the second event of type e_1	$\frac{3}{N+2}$	$\frac{1}{N+2}$	$\frac{1}{N+2}$
After the last event of type e_1	$\frac{c_1+1}{N+c_1}$	$\frac{1}{N+c_1}$	$\frac{1}{N+c_1}$
After the first event of type e_2	$\frac{c_1+1}{N+c_1+1}$	$\frac{2}{N+c_1+1}$	$\frac{1}{N+c_1+1}$
After the last event of type e_2	$\frac{c_1+1}{N+c_1+c_2}$	$\frac{c_2+1}{N+c_1+c_2}$	$\frac{1}{N+c_1+c_2}$
When the last event of type e_N arrives	$\frac{c_1}{N+\sum_1^N c_i-1}$	$\frac{c_2}{N+\sum_1^N c_i-1}$	$\frac{c_N}{N+\sum_1^N c_i-1}$
After the last event of type e_N	$\frac{c_1+1}{N+\sum_1^N c_i}$	$\frac{c_2+1}{N+\sum_1^N c_i}$	$\frac{c_N+1}{N+\sum_1^K c_i}$

Table 4: This table shows how the probabilities of event types are updated after they are sent. Each event type has prior probability $1/N$.

plying the probabilities of individual events, as shown below:

$$P(\mathcal{E}) = \frac{c_1! \cdot c_2! \cdot \ldots \cdot c_N!}{\frac{(\sum_{i=1}^{N} c_i + N - 1)!}{(N-1)!}} \tag{5}$$

The total cost of sending this message is negative base two logarithm of its probability. Thus, the total cost of sending the data is:

$$\begin{aligned} L(\mathcal{E}) &= -\sum_{i=1}^{N} \log c_i! + \log\Big(\sum_{i=1}^{N} c_i + N - 1\Big)! - \log\big(N-1\big)! \\ &= -\sum_{i=1}^{N} \log\Gamma(c_i + 1) + \log\Gamma\Big(\sum_{i=1}^{N} c_i + N\Big) - \log\Gamma\big(N\big) \\ &= -\sum_{i=1}^{N} \log\Gamma\big(c_i + \alpha_i\big) + \sum_{i=1}^{N} \log\Gamma\big(\alpha_i\big) \\ &\quad + \log\Gamma\left[\sum_{i=1}^{N} \big(c_i + \alpha_i\big)\right] - \log\Gamma\left[\sum_{i=1}^{N} \alpha_i\right] \end{aligned} \tag{6}$$

As seen in equation 6, the factorials can be replaced with gamma function, since $\Gamma(n) = (n-1)!$, if $n \in \mathbb{Z}^+$.

4.3 The Baseline Model

The baseline model encodes the aligned word pairs using prequential coding, directly in a way that is derived in the previous subsection 4.2. The algorithm used to complete the task consists of few steps. First the entire corpus is aligned randomly. Then one word pair at a time is aligned so that the cost of coding the complete data is greedily minimized immediately after each word pair.

The outline of the baseline algorithm is shown in in algorithm 1. The input of the algorithm 1 is a file that contains pairs of words in source and target languages. The number of word pairs is a subset of the complete data, since we include only complete word pairs; both languages must have a cognate. Typically, closely related languages share more common word forms than more distant languages.

Initially, we align all the given word pairs randomly. Basically this means random insertion of dots on source and target side, the only restriction is that a dot cannot be aligned with another dot. The random alignments are created with the help

Algorithm 1 The Baseline Model

Input: Corpus of n word pairs in source and target languages

Output: Globally optimal alignment of corpus

1: Align all word pairs *randomly*
2: Register the random alignments to the global count matrix
3: **repeat**
4: Read in the current alignment of a word pair $(\mathbf{s}_a, \mathbf{t}_a)$, $a \in [1, n]$.
5: De-register alignment of $(\mathbf{s}_a, \mathbf{t}_a)$ from the global count matrix.
6: Using current alignment counts, find the best *local* alignment for $(\mathbf{s}_a, \mathbf{t}_a)$.
7: Re-register the new alignment of $(\mathbf{s}_a, \mathbf{t}_a)$ to the global count matrix.
8: **until** The convergence of cost function

of dynamic programming procedure, where the cost in each step is random (see subsection 4.3.1 for more details). After all the word pairs are aligned once, the number of times each event occurs is updated to the global count matrix, that stores the event counts, as described in subsection 4.1.

There are different strategies to implement the search heuristics to find the globally optimal alignments. We have implemented *greedy* search strategy (subsection 4.3.2) and search using *simulated annealing* (subsection 4.3.3). Independent from search heuristics, the general ideas stay the same. Before trying to align a word pair, its contribution to the global count matrix is subtracted. We use *dynamic programming* to find the lowest cost alignment given the current parameter values from the global count matrix. After the new alignment is found, the counts of each event is added into the global count matrix so that the impact of new alignment is immediate in the value of cost function. The alignment steps are repeated until the convergence in cost using the following formula 7:

$$\begin{aligned} L_{base}(\mathcal{E}) &= -\sum_{e \in E} \log \Gamma\big(c(e) + \alpha(e)\big) + \sum_{e \in E} \log \Gamma\big(\alpha(e)\big) \\ &\quad + \log \Gamma \left[\sum_{e \in E} \big(c(e) + \alpha(e)\big) \right] - \log \Gamma \left[\sum_{e \in E} \alpha(e) \right] \end{aligned} \tag{7}$$

Here $c(e)$ is the number of times event e occurs in a complete alignment of the corpus, as stored in the global count matrix, and the $\alpha(e)$ are the Dirichlet Priors of the events. In the baseline model, we use uniform prior, where $\alpha(e) = 1$ for all events e.

4.3.1 Dynamic Programming

The general idea of *dynamic programming*, which was formulated by Bellman [Bel57], is to divide the problem into smaller subproblems and finally combine the solutions of the subproblems to get the solution to the complete problem. In dynamic programming, the subproblems are not independent. Each subproblem is solved only once and the solution is saved to be used later again to avoid the need of re-computation [Cor01].

The approach we use to perform the locally optimal alignment of a word pair, is similar to computing an edit distance between two strings, e.g. Levenshtein edit distance [Lev66], or the sequence alignment algorithms in genetics, e.g. Needleman-Wunsch algorithm [NeW70]. When trying to find the optimal alignment, we are allowed to do three different edit operations, i.e. substitution, insertion or deletion. Each such operation induces some cost that depends on the current parameter values, and the optimal alignment is the one for which the sum of the single operation costs is lowest.

Figure 4 shows a dynamic programming matrix that is used to align a source word $\mathbf{s} = [s_1, \ldots, s_n]$ with a target word $\mathbf{t} = [t_1, \ldots, t_m]$. In this matrix, each cell (s_i, t_j) corresponds to a partial alignment of words up to symbol i in source word and j in target word. The matrix is filled from top to bottom and from left to right. The different alignment possibilities correspond to different paths in the matrix starting from the upper-left corner and terminating in bottom-right corner.

Each cell of the matrix stores the cost of the most probable path so far, marked with $C(s_i, t_j)$ in cell (s_i, t_j). From each cell, it is only possible to continue the path

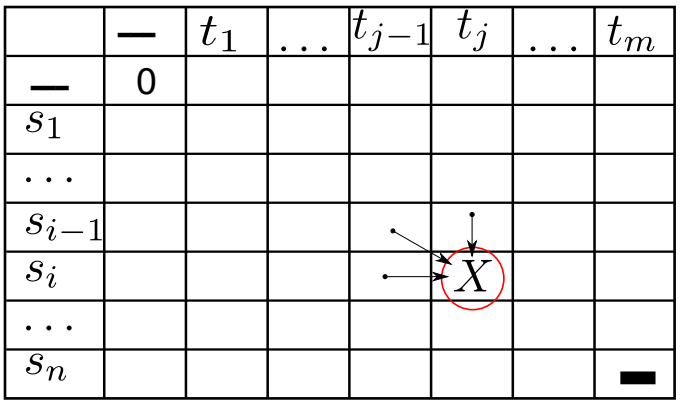

Figure 4: Dynamic programming matrix for computing the most probable alignment.

rightward (insertion of target symbol), downward (insertion of source symbol) or right and down (substitution of source and target symbol). The cost of each path is zero in the top-left corner, and the cost of the lowest cost path is stored into the the most bottom-right cell (s_n, t_m).

In figure 4 is a cell marked with X, where from source word $\mathbf{s}$ has been read off i symbols and j symbols from target word $\mathbf{t}$. The most probable path to cell X is one of the tree options that gives the lowest cost:

$$C(s_i, t_j) = \min \begin{cases} C(s_i, t_{j-1}) & +L(. : t_j) \\ C(s_{i-1}, t_j) & +L(s_i : .) \\ C(s_{i-1}, t_{j-1}) & +L(s_i : t_j) \end{cases} \tag{8}$$

The cost of path C is already computed for each cell up and left from cell (s_i, t_j) by dynamic programming. For any observed event ϵ, parameter $L(\epsilon)$ is the cost of the event. The cost can be computed from the change in the total code length when a new event ϵ is adjoined to the complete set of observed events $\mathcal{E}$ in count matrix, like shown in formula 9.

$$L(\epsilon) = \Delta L(\epsilon) = L\big(\mathcal{E} \cup \{\epsilon\}\big) - L(\mathcal{E}) \tag{9}$$

The tasks of finding the most probable path and the lowest cost path is equal; the parameter $P(\epsilon)$ which is the probability of an event ϵ can be expressed in terms of $L(\epsilon)$. The probability of an event ϵ of type e can explicitly be computed by combining equations 7 and 10, like shown in equation 11. In practice, we compute the most probable path using costs $L(\epsilon)$ instead of probabilities $P(\epsilon)$, due to the computational advantages.

$$P(\epsilon) = 2^{-\Delta L(\epsilon)} = \frac{2^{-L\left(\mathcal{E} \cup \{\epsilon\}\right)}}{2^{-L(\mathcal{E})}} \tag{10}$$

$$P(\epsilon) = \frac{c(e) + 1}{\sum_{e'} c(e') + |\mathcal{E}|} \tag{11}$$

4.3.2 Greedy Search

The first version of our baseline model is using greedy search heuristics for optimization of the decreasing cost function. At every phase of the search, the next state is chosen in a locally optimal way; after aligning a new word pair the parameter values are updated immediately. The algorithm is based on an assumption that the locally optimal choices will lead to globally optimal solution. The iterations are continued until the value of the global cost function is not decreasing any more. More discussion about greedy algorithms can be found for instance from [Cor01].

The advantage of the greedy search algorithm is its fast convergence. The disadvantage is that it indeed converges very easily to some local minimum. The result is relatively high total cost and appearance of events with very low counts. Because of this, we have chosen to use simulated annealing in search to prevent these problems, and after the convergence, use greedy steps if any are needed.

4.3.3 Simulated Annealing

Simulated annealing [KGV83, Čer85] is a probabilistic optimization method that is especially useful for approximating the global minimum of a cost function when there exists several local minima. Simulated annealing emulates a physical process of cooling a material into a minimum energy configuration [BeT93]. Simulated annealing is normally used in discrete finite search space. The general idea is that the cooling process starts from a high temperature; that is causing randomness in the selection of the following states. The temperature cools down as a function of time, and at the end of the simulation when the temperature is low, there is less randomness and the algorithm proceeds as a greedy search procedure. The purpose of the randomness is to prevent the algorithm to get stuck in some local optimum; this way it is possible to jump over those states.

Simulated annealing process starts from some initial state, $\epsilon_0 \in \mathcal{E}$. During each step, the algorithm moves to the next state, towards the minimum that is searched. The next state is chosen probabilistically from the set of candidates, which are neighboring states for the best candidate. That means the locally optimal solution is not always chosen. More formally, the simulated annealing process can be defined using the following terms and parameters:

- Cooling schedule is a function $\mathcal{T}(\tau) = \alpha\ \mathcal{T}(\tau - 1)$, where τ is time. $\mathcal{T}(\tau)$ is

also called the temperature at time τ.

- The initial temperature $\mathcal{T}(0)$ is initially quite high and is problem specific. We use values $\mathcal{T}(0) = 50$ or $\mathcal{T}(0) = 100$.
- Cooling parameter $\alpha \in \mathbb{R}$, where $0 < \alpha \leq 1$. In our experiments, the parameter value is typically $0.99 \leq \alpha \leq 0.995$.

We use the simulated annealing during the dynamic programming process. Normally, at each cell of the dynamic programming matrix, the next state is determined by the most probable path through the matrix that far. In the 1x1 alignment, there are always 3 possible states (candidate events) to come from, and the candidate with lowest cost is chosen. In case of a tie, the candidate is chosen randomly. When we involve the simulated annealing, the candidate selection becomes a probabilistic process. Let ϵ_b be the best possible candidate event with the lowest cost $L(\epsilon_b)$ and ϵ_i any other possible candidate event with cost $L(\epsilon_i)$. For each possible event ϵ_i, $i \neq b$, is computed the following probability at time τ:

$$P(\epsilon_i, \mathcal{T}(\tau)) = \exp\left[\frac{-1}{\mathcal{T}(\tau)}(L(\epsilon_i) - L(\epsilon_b))\right] \tag{12}$$

Let $P_{\texttt{rand}}$ be a random probability, where $0 \leq P_{\texttt{rand}} < 1$. The next possible state is chosen randomly from a set of events, in which belong the events ϵ_b and all ϵ_i for which $P(\epsilon_i, \mathcal{T}(\tau)) > P_{\texttt{rand}}$. When the temperature $\mathcal{T}(\tau) \to 0$, the probability $P(\epsilon_i, \mathcal{T}(\tau)) \to 0$ and it is less and less probable that other but the lowest cost candidate is chosen.

The cooling schedule is updated every time all the word pairs in data are aligned. The process is repeated until the convergence of the cost function or the temperature reaches some predefined value, e.g. $\alpha = 0.00001$.

4.4 Two-Part Code Model

The two-part coding is a more clever way to encode the data, and performing the search with respect to the two-part code cost function yields lower cost and also better alignments. The only difference in two-part code model to the baseline model is the reformulated cost function, the algorithm stays the same.

The nature of the phenomenon of sound change is that the number of changes that develop from a common ancestor language, is small. This doesn't mean that sounds

should always align with themselves, but there should be regularity that leads to a situation, where only a small proportion of all possible events in $\mathcal{E}$ will ever occur. This should be seen from the data, and we expect the event matrix to be *sparse* to reflect this fact.

With the help of two-part code, we have to send information only about the events that have non-zero counts; events that have actually been observed. Two-part code consist of two parts: the first part (CB) is the *codebook* that encodes the parameters of the models and the second part ($\mathcal{E}|CB$) encodes all the instances of the events (the data), with the help of the codebook. The two-part code cost function, the total code length we try to minimize, is shown in formula 13:

$$\begin{aligned} L_{tpc}(\mathcal{E}) &= L_{tpc}(CB) + L_{tpc}(\mathcal{E}|CB) \\ L_{tpc}(CB) &= \log(|E|+1) + \log \binom{|E|}{|E^+|} \\ L_{tpc}(\mathcal{E}|CB) &= -\sum_{e \in E^+} \log \Gamma\big(c(e)+1\big) + \log \Gamma \left[\sum_{e \in E^+} \big(c(e)+1\big) \right] - \log \Gamma(|E^+|) \end{aligned} \tag{13}$$

The cost of the codebook $L(CB)$ tells how many bits are needed to encode the observed events. The first term $\log(|E|+1)$ in the codebook is used to measure the cost of coding the number of non-zero event types. The second term $\log \binom{|E|}{|E^+|}$ is the cost of specifying which subset $E^+ \subset E$ of all the possible event types have non-zero counts. The cost of the second part, $L(\mathcal{E}|CB)$, is derived using Bayesian marginal likelihood. Similarly to the baseline model, we have set all priors $\alpha(e)$ to one.

In addition to the expectation of sparse data, we can also assume that the the different alignment types behave differently. We can define four different *kinds* K of 1x1 symbol alignment, where source symbol $s \in S$ and target symbol $t \in T$:

$$K_{1x1} = \Big\{ \ (s:t), \quad (s:.), \quad (.:t), \quad (\#:\#) \ \Big\} \tag{14}$$

The kinds allow us to encode their occurrences separately in the codebook of the two-part code. If the assumption about their different behaviour is correct, this modification results in lower cost. The cost function with reformulated codebook is shown below:

$$
\begin{aligned}
L_{kinds}(\mathcal{E}) &= L_{kinds}(CB) + L_{kinds}(\mathcal{E}|CB) \qquad (15)\\
L_{kinds}(CB) &= \sum_{k \in K} \left[\log(N_k + 1) + \log \binom{N_k}{N_k^+} \right] \\
L_{kinds}(\mathcal{E}|CB) &= -\sum_{e \in E} \log \Gamma\big(c(e) + 1\big) + \log \Gamma \left[\sum_{e \in E} \big(c(e) + 1\big) \right] - \log \Gamma(|E|)
\end{aligned}
$$

Here N_k is the number of possible event types of kind $k \in K$ and N_k^+ is the corresponding number of such event types actually observed in the alignment; by definition we get $\sum_k N_k \equiv |E|$.

As already mentioned, we use algorithm 1 also for the two-part code model. Because the cost function is changed, also the path cost computation is slightly different in dynamic programming phase. Like in the baseline model, the parameter values $P(\epsilon)$ for every observed event $\epsilon \in \mathcal{E}$ are still computed from the change of code length, like defined in equation 10. If the event type e of kind k is already observed, the count of the event type $c(e) > 0$ and we can use the update formula of equation 11. It is also possible that we observe an event of completely new kind. In this case, it is more expensive to add this event. The probability of adding a new event of kind k is then (equation 16):

$$
P(\epsilon) = \frac{1}{\sum_{e'} c(e') + |E|} \cdot \frac{|E|}{\sum_{e'} c(e') + |E| + 1} \cdot \frac{N_k^+ + 1}{N_k - N_k^+} \qquad (16)
$$

4.5 Multiple Symbol Alignment

The two-part coding is further utilized in multiple symbol alignment. The 2x2 model that is described in this section, is an extension of the baseline and two-part code models and it is designed to find correspondences of up to two symbols on both source and target side. The possible symbol correspondences are then one-to-one, one-to-two, two-to-one and two-to-two.

Like in 1x1 two-part code model with kinds, we start by introducing a set of possible kinds. The equation 17 shows the set of all different kinds we have defined. The source symbols are $s, s' \in S$, target symbols $t, t' \in T$, symbol ^ denotes the beginning of a word and \$ symbolizes the end of a word. We also model deletions and insertions (a dot on source or target side) with their own kinds.

$$K_{2x2} = \left\{ \begin{array}{llll} (\hat{} : \hat{}), & (\hat{} : \hat{}t), & (\hat{}s : \hat{}), & (\hat{}s : \hat{}t), \\ (. : t), & (s : .), & (ss' : .), & (. : tt'), \\ (s : t), & (ss' : t), & (s : tt'), & (ss' : tt'), \\ (\$: \$), & (\$: t\$), & (s\$: \$), & (s\$: t\$) \end{array} \right\} \tag{17}$$

There are now 16 different kinds defined, whereas the number of kinds in the 1x1 model was 4. Also the size of the event space in now larger since for each new kind there is a certain number of additional event types and instances. However, we can expect the global count matrix to be very sparse, since only a small proportion of the two-to-two event types are supposed to occur. The objective function is the same as in formula 15.

The dynamic programming algorithm in the re-alignment step of algorithm 1 has to be adjusted, because of two reasons. The first reason is the addition of the word boundary symbols, the second reason is the grown number of possible ways to move in the dynamic programming matrix.

Figure 5 shows an example of dynamic programming matrix for Finnish-Estonian word pair *pyörä-pöör*. Like shown in equation 17, the both word boundary symbols (ˆ and \$), can only be aligned with a symbol or symbol pair that contains themselves and that has to be taken into account when looking for the lowest cost path through the matrix. The border cells that are filled with gray color are now impossible, and no path can go through them.

Another difference is that to each cell of the dynamic programming matrix leads now maximum 8 different paths since it is now possible to read off up to two symbols at a time. The cost of each step is computed as in 1x1 case, using formulas 11 and 16. In figure 5 are shown the 4 possible ways to align the ends of the given word pair. Arrow #1 leads to alignment that ends with *ä*\$-*r*\$, arrow #2 to \$-*r*\$, arrow #3 to *ä*\$-\$ and arrow #4 to \$-\$.

The red arrows show the lowest cost path that can be backtracked from the last cell of the matrix to the beginning. This step is not different from the same task in 1x1 models. The resulting path in figure 5 is:

$$\begin{array}{cccc} \hat{}p & y\ddot{o} & r & \ddot{a}\$ \\ | & | & | & | \\ \hat{}p & \ddot{o}\ddot{o} & r & \$ \end{array}$$

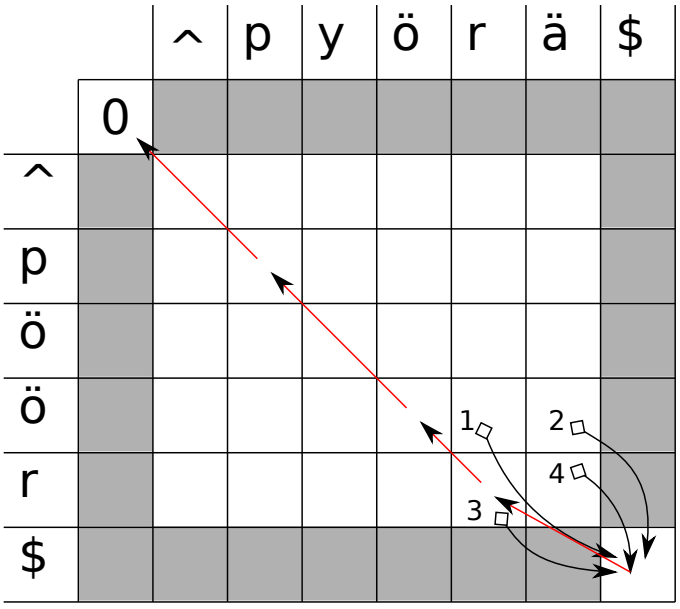

Figure 5: A dynamic programming matrix in multiple symbols alignment. The shadowed cells are impossible to enter. There is also shown 4 different ways to enter the last cell of the matrix.

4.6 3-Dimensional model

All the models described this far align languages pairwise. The learning of 1-to-1 sound correspondences is easy to extend to three dimensions. The differences and modifications to the two language models are described in this subsection.

The 3-dimensional alignment was described in subsection 4.1. As defined there, we use marginal approach to keep track of the alignment counts. To be able to get more data for learning, we accept to the input data all cognate sets that contain words from at least two of the languages to be aligned. Because of the incomplete examples, we need to define two different ways to compute the code length of an alignment. If we have a complete alignment, the cost of an event is computed as the sum of the pairwise event costs, based on the three marginal count matrices. In case of an incomplete example, the cost is obtained using the corresponding marginal matrix only. The cost of 3-dimensional alignment in each case is formulated below:

$$
\begin{aligned}
L_{3D}(s:t:v) &= L_{ST}(s:t) + L_{SV}(s:v) + L_{TV}(t:v) \\
L_{3D}(s:t:-) &= L_{ST}(s:t) \\
L_{3D}(s:-:v) &= L_{SV}(s:v) \\
L_{3D}(-:t:v) &= L_{TV}(t:v)
\end{aligned} \tag{18}
$$

The pairwise alignment costs are computed using two-part coding, either using equation 13 or equation 15.

The algorithm follows the baseline algorithm 1. The dynamic programming phase is different for complete and incomplete examples. For incomplete examples, we use the same procedure as in two dimensional case, see section 4.3.1. For complete examples, the dynamic programming matrix is extended to 3 dimensions. When in 2-dimensional case, the task was to compute the cost of a path into a cell (s_i, t_j) for each i, j, in 3-D world the same task is extended to find the cost of a path into a cell (s_i, t_j, v_k), where k is the third dimension. The minimum cost path is computed using the 3-dimensional event costs $L_{3D}(s:t:v)$. The number of possible paths to each cell grows from 3 to 7 as illustrated in figure 6.

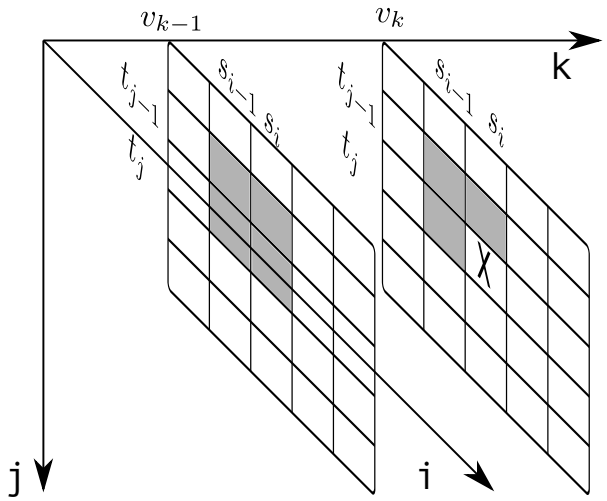

Figure 6: A part of dynamic programming matrix in three dimensions. Because of the third dimension k, there are now 4 additional cells on plane $k-1$ from which to move to cell (s_i, t_j, v_k), which is marked with X in figure.

4.7 Context Based Model

Context based model is a two-dimensional model, that takes context, or the environment of an alignment into account. For one word pair, the awareness of environment means that cost of an alignment depends on the previous alignment decisions. Differently to the previously introduced models, an alignment is not between symbol pairs, but on the level of individual features of the symbols. The conversion of symbols to the feature space is described in section 3.2. The features are encoded using decision trees that are used to split the data in the most informative way.

4.7.1 Contexts

More formally, when encoding a single feature f, *context* is a triplet $(\mathcal{L}, \mathcal{P}, \mathcal{F}\ [X])$ that is some environment of f. Level $\mathcal{L}$ is source or target level. Position $\mathcal{P}$ is in relation to the position of f. A list of positions that are possible to query are listed in table 5, it is only possible to query contexts that are already encoded. Feature $\mathcal{F}$ is one of the features from table 3 and optional attribute X is one of the values of that feature. A possible context is e.g. (`SOURCE`, `PREVIOUS POSITION`, `TYPE`), that asks what is the value of the `TYPE` feature in previous position on source level. That context has four possible values, the `TYPE` can be a dot, a vowel, a consonant or a word boundary. Binary questions use optional attribute `X`. This kind of query could be (`SOURCE`, `PREVIOUS POSITION`, `TYPE V`): is the type of the symbol in previous position on source level a vowel.

The set of possible contexts for a feature f is large. There are 2 levels: source and target, 8 positions, 10 features and on average 4 feature values for each feature. This sums up to 160 general queries plus 640 binary queries. The maximum number of (theoretically) possible contexts is then 800. In practice the number is smaller, since we do not define redundant contexts, e.g. if the position is `PREVIOUS VOWEL`, it is not useful to query the consonant features. Since the features are encoded in fixed order, the number of contexts also depends on the position of f: it is largest for the last feature of the target symbol in order and smallest for the first feature of the source symbol. The summary is that the number of contexts varies from feature to feature, but it is a fixed, predetermined number.

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

Table 5: All possible positions for the context model

4.7.2 Decision Trees

The alignments produced by context model are encoded using prequential coding. We use *decision trees* in order to determine the cost of coding the feature f. The theory of decision trees can be studied from textbooks, e.g. [RuN10]. The coding of decision trees is discussed e.g. in [QuR89, WaP93].

The model consists of total 20 decision trees, since we build one tree for each feature and for each level. That is 5 consonant feature trees, 4 vowel feature trees and 1 type feature tree, on both levels. We store all the data in these trees, each tree containing information about the corresponding feature. The goal of tree building is to learn regularities from the data and consequently reduce the entropy of the data, to further reduce the cost of coding.

The feature trees are encoded in a fixed order and all the source trees are encoded before the target trees. Here it is important to notice one main difference to the context-free models, where the events are coded jointly. Thus those models are symmetrical with respect to the source and target languages. This is not the case with the context model, where the cost of coding a feature is conditioned by its context, and the set of possible contexts is different on source and target levels.

In order to initialize the trees, we need a random alignment, created same way as in baseline model (see section 4.3). After the initialization, we build the trees, one by one. The first step is always to build the *root node* of the feature tree. The root node contains a *count matrix*, where we store a distribution over the values of the feature. In figure 7 a. is shown the root node of `HORIZONTAL/TARGET` tree: it is a `TARGET` level tree for `HORIZONTAL` vowel feature. The possible values of `HORIZONTAL` feature are front (F), medium (M) and back (B). To fill in the count matrix, we

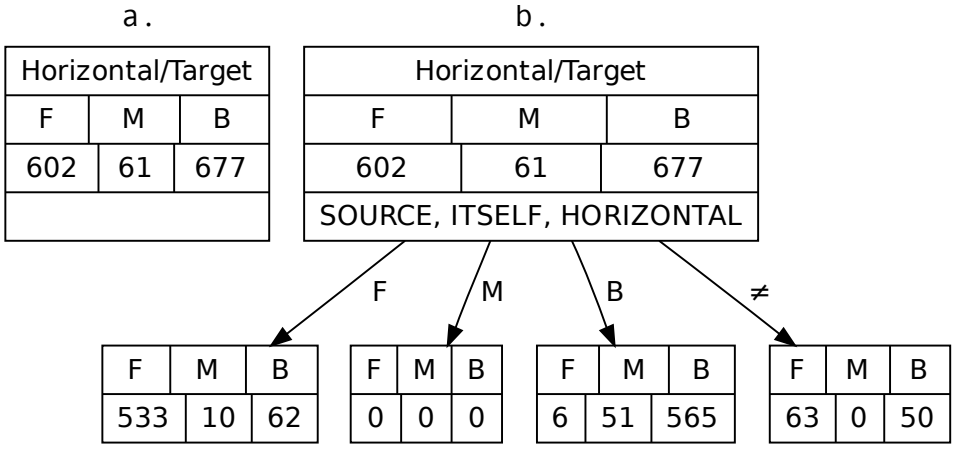

Figure 7: A decision tree for HORIZONTAL feature on TARGET level. The best candidate to split the root node is (SOURCE, ITSELF, HORIZONTAL), which gives the greatest reduction to the entropy of the count matrix in the root node.

count the number of times each value occurs in vowels of the target language. The data in example is Estonian words that have a Finnish cognate.

The next step is to look for a way to split the root node so that the entropy of the count matrix is minimized. The set of candidates that can be used for splitting is any of possible contexts $(\mathcal{L}, \mathcal{P}, \mathcal{F}\ [X])$. The best candidate is always the one that produces the greatest reduction in the total tree cost. For each candidate, the split is conditioned on all possible values of $\mathcal{F}$ and if needed, on values that indicate non-matching values, which include word boundaries (#) and wrong feature type ($\neq$). Each of the values results in a new branch diverging from the root node, ending up to a child node that contains an updated count matrix. Figure 7 b. shows the resulting tree if the splitting is conditioned on candidate (SOURCE, ITSELF, HORIZONTAL).

The complete tree is built recursively, from top to bottom, the aim is to split all nodes as long as it improves the total cost. The number of possible candidates is decreasing down the tree, after a candidate is once used, it cannot be further utilized on lower levels of the tree. At each node all the remaining candidates are gone through exhaustively, and the best split is chosen in a greedy manner. The recursion ends when the total tree cost stops decreasing.

We use two-part coding to compute the total cost of a tree. The first part is the *model cost* $L_{tree}(M)$, that is used to encode the structure of the decision tree. The cost of a model M is used to encode all the decisions made to build the tree. The

following model cost function, which is divided in two parts, adds up in every node of the tree:

$$L_{node}(M) = \begin{cases} 1 + \log(|Ctx_f^l| - d) & \text{if split} \\ 1 & \text{if no split} \end{cases} \tag{19}$$

The explanation is that if we do not split a node, we need 1 bit to encode that decision. If there is a split, that costs 1 bit to code the decision and $\log(|Ctx_f^l| - d)$ bits to tell which of the candidates we used to perform the split. Here $|Ctx_f^l|$ is the number of candidates of feature f on level l, and d is the depth of the tree, starting from $d = 0$ at root node. The model cost of a tree is $L_{tree}(M) = \sum_{node} L_{node}(M)$.

The second part of the two-part code is the *cost of the data given the model*, $L_{tree}(\mathcal{E}|M)$. It consists of the sum of the costs of the count matrices in the *leaf nodes* of the tree. The cost of one count matrix can be computed using the prequential formula 7.

4.7.3 Context Based Alignment

We use dynamic programming to find the locally best alignment for a word pair. We start from random alignments, and build all the trees according to the random data. In context based alignment, each symbol alignment $(s_i : t_j) \in S_\cdot \times T_\cdot$ is represented feature-wise and each feature is encoded separately. Thus, all feature in s_i and t contributes one event into the corresponding tree.

Before performing the re-alignment of a word pair w_k using dynamic programming, we must remove the current contribution of events in w_k from the feature trees. At this step, the structure of the trees remains untouched. Instead, for each feature, we modify the data distribution in the leaf node in which the event was stored. This leaf node is determined according to the context of the symbol at hand.

In figure 8 is shown the `TYPE/SOURCE` tree that is always the first tree in order to encode, whether s_i is a dot, a vowel, a consonant or a word boundary symbol. We start from the root node, then query the context that splits the root node and continue until a leaf node is reached. In the example of figure 8 the first query is `(SOURCE, PREVIOUS SYMBOL, ROUNDED)`. For the first symbol s_0 in w_k, we follow the edge marked with #, since s_0 is preceded only by the word boundary symbol #. If the source symbol s_0 is e.g. a consonant, we look at the box marked with `C` in the count matrix of the leaf node, and subtract 1 from the value of `C`. This procedure is

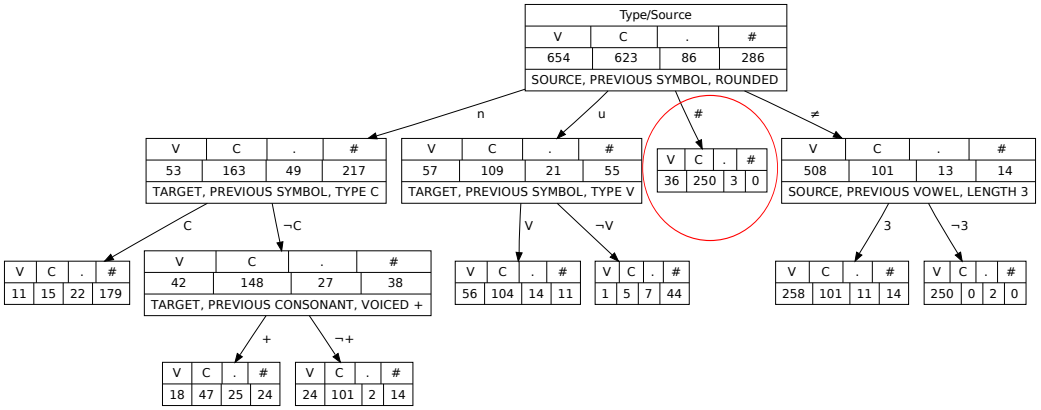

Figure 8: A decision tree for the TYPE feature on SOURCE level for Finnish-Hungarian alignment. At the beginning of a word, the # edge is always chosen, since the preceding symbol can be nothing but a word boundary.

repeated for all the features of all s_i and t_j in w_k.

The next step is the dynamic programming, which proceeds as in the baseline algorithm and is described in section 4.3.1. The cost of adding a symbol alignment $(s_i : t_j)$ is now computed as a sum of costs of the feature-wise events in $(s_i : t_j)$. In order to compute the cost of adding a feature-wise event, we have to trace the leaf nodes in which the events are added. The leaf nodes are found similarly to the previous example of figure 8. For one feature-wise event, the change in total code length is cost of adding the event to the leaf node, and it can be computed using the equation 11. The lowest cost path through the dynamic programming matrix is the new resulting alignment of w_k, and the participating events are updated into the trees.

Finally, when all the word pairs are re-aligned, the trees are rebuilt. This is the phase where it is possible to learn new rules, since the structure of the trees may be changed. The total cost is computed after the rebuilding, and the steps are repeated until the cost cannot be improved.

4.7.4 Convergence of Context Model

The greedy search procedure (as described in section 4.3.2) does not help the context based alignment to converge to a globally optimal state. As a consequence, the simulated annealing approach (section 4.3.3) is adapted, with some modifications. The standard simulated annealing requires very slow schooling schedule to work with

the context model. Since the trees have to be rebuilt after every complete alignment iteration, the execution of the algorithm slows down. The study of context model is still an ongoing project, and this far we have tried several search variants to compromise the slow execution times and goodness of the found optimum. The following variants are all used, they are listed here with their pros and cons:

1. **Standard simulated annealing**, using cooling parameter $\alpha = 0.995$. This variant results in a slow running time, but relatively low global cost. The learned rules of correspondence often lead to *skew alignments*, where the corresponding symbols of source and target languages do not align directly but with a shift off up to 3 symbols. This phenomenon does not necessarily make the rules of correspondence learned any worse, but they do make the evaluation a challenging task.

2. **Pipeline approach**, where the initial alignment given for the context model, is a result of converged two-part code model, and the decision trees are built based on that. The greedy search strategy is repeated, until the cost cannot be reduced; typically only for a few iterations. This approach is very fast, but there is danger, that the optimum found by the context-free model prevents from finding a better optimum.

3. **Zero-depth convergence** method combines simulated annealing and the greedy search. During the simulated annealing, only the root nodes of the source level trees are built and and target level trees may query only the candidates in `ITSELF` position. After the convergence, all the limitations of tree building are removed, and the rules are learned using the greedy search strategy. This method leads to straight alignments and relatively low cost, but it has more limited search space than the plain simulated annealing approach.

4. **Infinite-depth convergence** limits the number of candidates that can be used to split the nodes during the simulated annealing phase. The root node in any tree can be split only by a candidate in some already encoded `ITSELF` position. This variant was designed to prevent the skewness of the alignments, but did not work that well on this task. It is still usable, since it is less limited than the zero-depth convergence method, and converges faster than the standard simulated annealing approach.

5 Experiments and Results

In this chapter is discussion about the experiments we have implemented to evaluate the capabilities of the presented models. The evaluation task is challenging, mainly because there is no direct answer to the question of how the words should ideally align. For that purpose, we would need *a gold standard*, which would define the sound changes that actually have happened during the language evolution. If the gold standard was available, it would be easy to compare the alignments produced by our models to the expected alignments. This way we could measure the accuracy of our models exactly. Unfortunately, there exists no gold standard for the Uralic data we are experimenting. However, there are some indirect ways to do the evaluations, these methods are described in the following sections.

5.1 Code Length and Compression Rates

The evaluations in this subsection mainly aim to differ the compression power of our models. This is an indirect way to measure the capabilities of the models to find regularities from the data. Information theoretic methods to measure linguistic complexity are represented e.g. by Patrick Juola in [Juo98]. In his work, the standard compression techniques (like Lempel-Ziv algorithm [ZiL77]) are used to compress language samples to approximate their Kolmogorov complexity and to measure the amount of information of the full corpora from which the samples originate. The same test is used also in [KSL06].

The initial step in the evaluation is very simple and fast; a comparison of the code length of the alignments when different models are used. Since all our methods use code length as an optimization criteria, it is natural to assume that the better the model is, the lower is the code length. Of course, it is possible to build a model that is wasting bits while doing excellent job with alignments, but on the other hand, a very badly performing model cannot achieve very low code length. In any case, the performance of the models cannot be judged only by the ranking in a code length comparison. The purpose of the code length comparison is mainly to:

1. Compare the compression power of our different models to help us to select the best model from a set of different model variants.

2. Detect possible problems with the search heuristics (e.g. if a very restricted model finds lower optimum than more loosely restricted).

	Baseline model	1x1 model	2x2 model	Context model
est-fin	40658.47	21691.78	19701.03	**18952.91**
est-khn	15817.35	11123.26	10507.24	**10234.46**
est-kom	17326.66	11824.50	11005.27	**10727.84**
est-man	14421.26	10390.48	9896.18	**9633.99**
est-mar	16944.73	11590.45	**10920.71**	11062.66
est-mrd	23142.90	15124.70	14246.11	**13872.66**
est-saa	21015.98	14116.95	**12721.54**	12799.43
est-udm	15705.50	10955.40	10203.54	**10053.56**
est-ugr	12776.70	9219.29	**8605.15**	8709.49

Table 6: Estonian data is aligned with 9 other Finno-Ugric languages using four different models. The data sets are cognate sets from the StarLing database. For each language pair, the lowest code length (in bits) is marked with boldface font. Notice that the code lengths are not normalized here and the costs are not comparable row-wise.

3. Compare the differences in code length for different language pairs. In order to do that properly, the inter-language distances need to be normalized. This is done in section 5.2. Normalization is needed because there is varying amount of data for different language pairs. Typically, the closer relationship the language pair has, the more data they have in common.

In table 6 is compared different models using the code length of certain language pairs. The context model has the best compression power for most of the language pairs. Baseline model, which is using the simple prequential coding, performs very badly in this comparison. 1x1 model is using two-part code, and those results are already comparable to the more sophisticated 2x2 model and context model.

In figure 9, the compression power of the models is compared against standard compression applications, *gzip* and *bzip2*. Gzip is a lossless data compression algorithm which is based on Lempel-Ziv algorithm [ZiL77] and Huffman coding [Huf52]. Bzip2 is a more efficient compression algorithm that is using Burrows-Wheeler transform [BuW94]. Both of these algorithms are general purpose algorithms, and there certainly exist more efficient compressors. However, this illustration gives an interesting view to the compression capability of our algorithms in this specific task. Learning about the vertical correspondences helps our models to achieve better compression rates. The data is from SSA data set and the biggest data sample contains over 3000 Finnish-Estonian word pairs. The data samples are given to the compressors in files

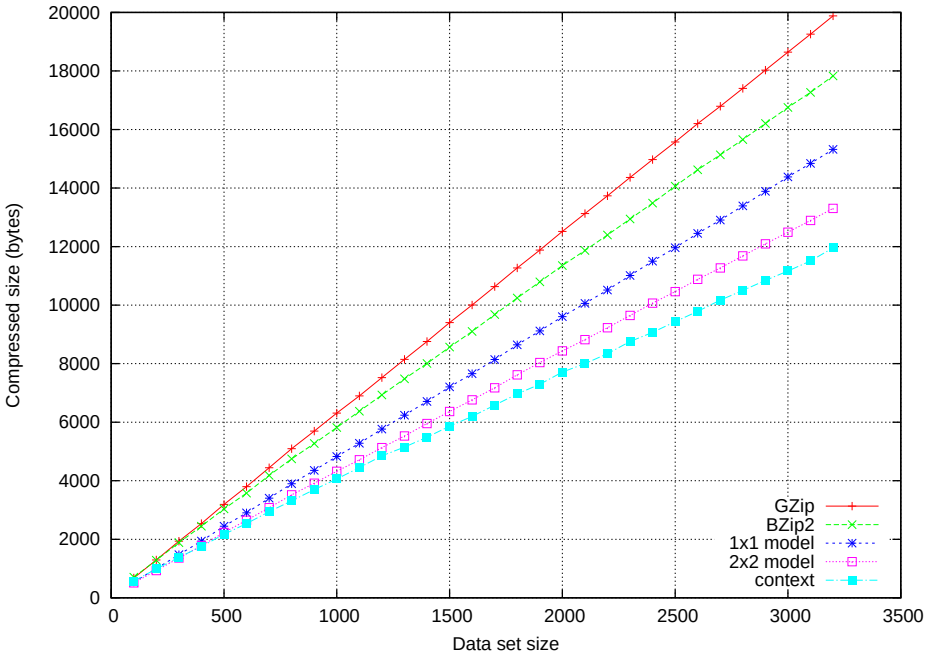

Figure 9: A comparison on compression power of gzip, bzip2, 1x1 model, 2x2 model and context model. The compression power increases in this order. The data samples are Finnish-Estonian word pairs from SSA database.

in which there is one word per line. After the data is processed by the algorithms, the compressed file size is checked. We get the compression power of our methods directly from the code length computed during the execution of those algorithms.

5.2 Language Distances

One of our goals is to determine the etymological relationships of the Finno-Ugric languages. This is a step towards reconstruction of the proto-forms of the extincted languages of the language family. We can use our models to measure inter-language distances. The first step in this task is to align all language pairs in StarLing data set. Since we have 10 languages, we get the code length for 100 language pairs.

Since the language distances are not directly comparable, they have to be normalized. For measuring the similarity between languages, we can use Cilibrasi and Vitanyi's method [CiV05] to compute the *normalized compression distance, NCD*.

	fin	*khn*	*kom*	*man*	*mar*	*mrd*	*saa*	*udm*	*ugr*
est	0.37	0.75	0.70	0.74	0.71	0.69	0.49	0.70	0.66
fin		0.74	0.72	0.76	0.70	0.67	0.43	0.693	0.67
khn			0.73	0.70	0.69	0.70	0.62	0.68	0.75
kom				0.78	0.68	0.62	0.55	0.50	0.70
man					0.72	0.63	0.61	0.76	0.76
mar						0.59	0.48	0.70	0.59
mrd							0.48	0.60	0.62
saa								0.53	0.56
udm									0.70

Table 7: Pairwise normalized compression distances for Finno-Ugric sub-family of Uralic, in StarLing data.

This similarity metric can be used in all application areas and it is independent from the type of compressor used. The underlying idea is again that the sizes of compressed files can be used to approximate their Kolmogorov Complexity [LiV94].

The normalized compression distance $\delta(a, b)$ is defined between languages a and b as follows:

$$\delta(a,b) = \frac{C(a,b) - \min(C(a,a), C(b,b))}{\max(C(a,a), C(b,b))} \tag{20}$$

where $0 < \delta < 1$ and $C(a, b)$ is the compression cost between the two languages. Cost $C(a, a)$ is for a monolingual alignment; a language aligned with itself. According to the equation, it is possible that the relation may be asymmetric. We have averaged the pairwise distances to be symmetric. The pairwise compression distances, computed using a context model, are shown in table 7.

The pairwise compression distances can be used to induce a phylogenetic tree that shows the relations of the languages. We used *UPGMA* algorithm [Mur84] to draw the tree. UPGMA (Unweighted Pair Group Method with Arithmetic mean) is a hierarchical clustering method that is used for the creation of phylogenetic trees. In the construction, the algorithm uses the distance matrix. Basically, at each step, UPGMA combines two nearest clusters to form a higher-level cluster. Once a cluster consists from more than one members, the distance is computed by taking the mean distance between all elements of each cluster. If there are clusters Q and R, the mean distance between these clusters is:

$$\frac{1}{|Q| \cdot |R|} \sum_{q \in Q} \sum_{r \in R} d(q, r) \tag{21}$$

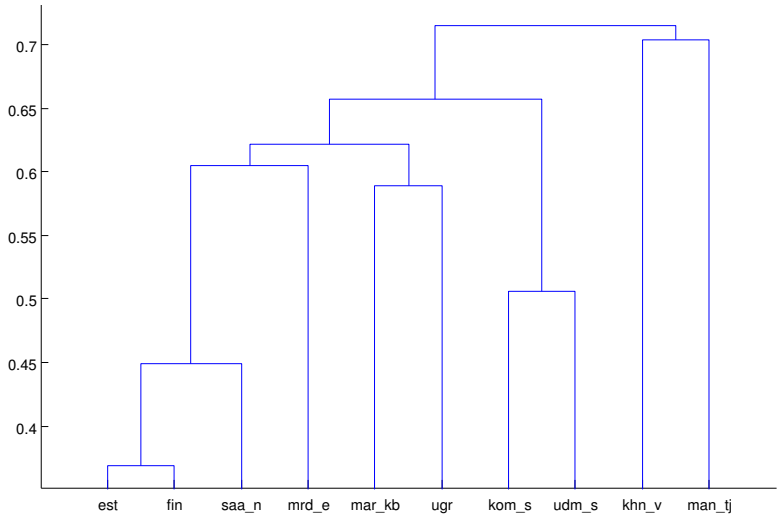

Figure 10: Finno-Ugric tree induced by NCD.

The UPGMA algorithm is straight-forward to use, but it is not an ideal algorithm in many cases. A tree constructed using the UPGMA is shown in figure 10, and it is using the precomputed distances of the table 7. The UPGMA-tree can be compared to a tree built by linguists [Ant89] in figure 1. The reconstruction captures the relations of closely related languages very well, but is no that successful to model the more distant relations. One another option to consider for tree construction is for instance Cilibrasi's CompLearn Algorithm [CiV11].

5.3 Quality of Alignments

Even though we can not evaluate the goodness of the obtained alignments quantitatively, it is possible to perform a qualitative evaluation. The approach here is to check what kind of *recurrent sound correspondences* the model of interest can find and compare them to well-known correspondences that can be studied from the linguistic handbooks, e.g., [Lyt73, Sin97].

In figure 11 there is a two-dimensional global count matrix that models the alignment of Estonian with Finnish. This example matrix is produced by the two-part code model, as described in chapter 4.4 and later referred in this chapter as 1x1 model. The row labels of the matrix are from the Estonian alphabet, column labels from the Finnish alphabet. The count matrix describes the complete final alignment. In this visualization, the actual alignment count between corresponding symbol pair in the corpus is replaced with a ball of which size is proportional to this number.

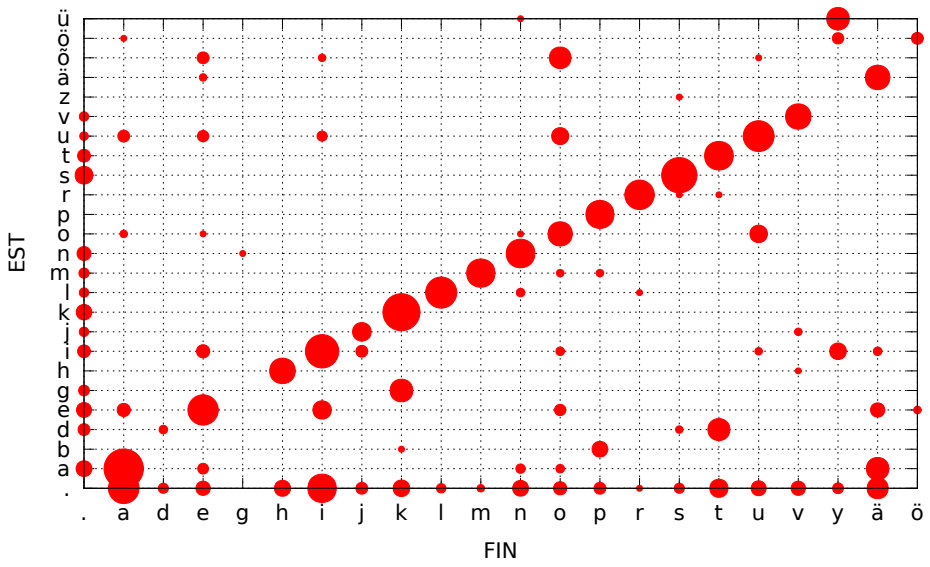

Figure 11: Alignment matrix for Finnish-Estonian data using the 1x1 model.

Finnish and Estonian are very closely related languages, so the alphabets are also very close to each other. However, it is important to notice that the model we use knows nothing about the symbols themselves. We could be aligning just numbers or letters from completely different alphabets, e.g. Cyrillic and Latin. Even though Finnish *a* looks similar to Estonian *a* for us in this example that is not the case for the model since it codes the source and target level symbols separately, independently from each other. Given this, model finds alignment from which it is easy to see that there is a very simple kind of correspondence between similar looking symbols, e.g. Finnish *a* and Estonian *a*.

As a consequence, there is a clear emphasis of the regular alignments on the diagonal of the matrix. In general, the clearer the diagonal appears, the less ambiguous are the alignments and the lower the entropy of the matrix is. In other words, we consider the quality of alignments to be higher the lower the entropy in the alignment matrix is. If there exists many ambiguous alignments, we hope to find an explanation, a rule that describes them as completely as possible.

From the example of figure 11 can be seen that there are regular deletions of Finnish vowels *a*, *i* and *ä*. This phenomenon is due to fact that word-final vowels are often deleted in Finnish if compared to Estonian. Another example is Estonian vowel *o*, which sometimes corresponds to Finnish vowel *o*, sometimes to Finnish *u*. This correspondence cannot be explained by this model only, we need a more complex

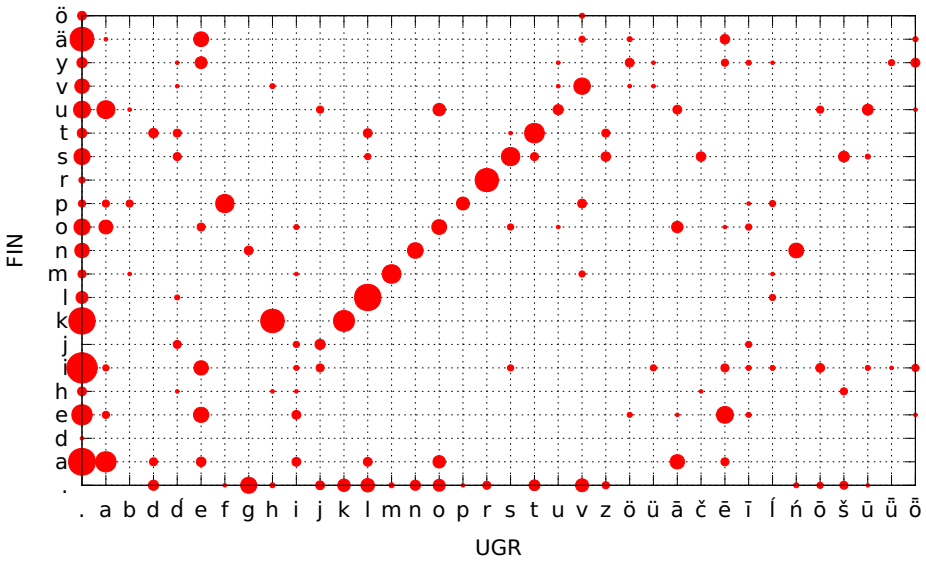

Figure 12: Alignment matrix for Hungarian-Finnish data using the 1x1 model.

rule to infer it.

In figure 12 is shown an alignment matrix for a language pair Hungarian-Finnish. These two languages are very distant relatives, and this fact can also be seen from the matrix visualization. Compared to figure 11, there is still a diagonal of strong sound correspondences, but much more ambiguous events that are difficult to explain with the help of the matrix only.

It is also possible to visualize the 3-dimensional alignment matrix, to learn more about the sound correspondences inside the language family. In figure 13 is shown an alignment matrix that is a result of aligning Mordvin, Estonian and Finnish languages simultaneously. This view gives more information about the rules in general.

5.4 Rules of Correspondence

The simple correspondences described in previous section 5.3 are interesting as a sanity check of the performance of our methods, but it is even more interesting to model more complex rules of correspondence and compare them to manually found rules, which can be expressed in terms of *two-level morphology* by Kimmo Koskenniemi [Kos83]. Koskenniemi's two-level model consists of two components, which are a *rules component* and a *lexical component*. The rules component contains

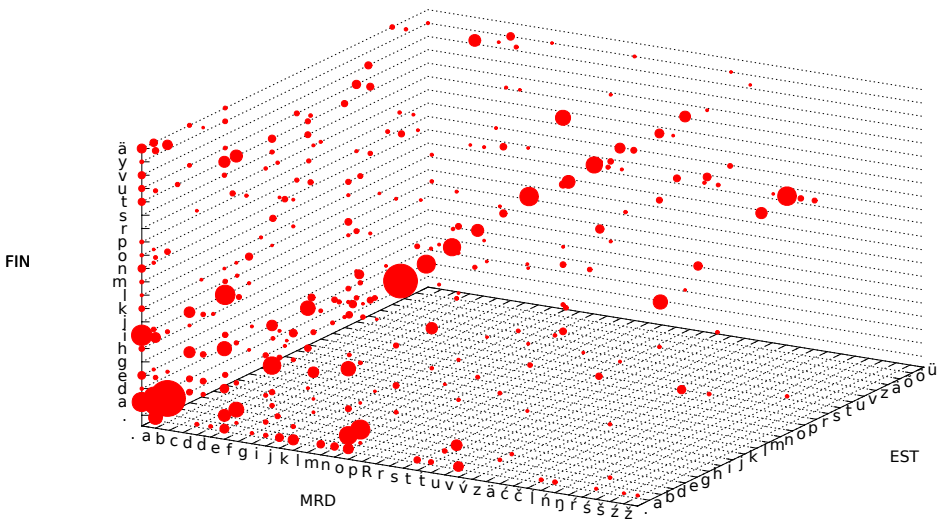

Figure 13: 3-D alignment matrix for Mordvin-Estonian-Finnish data.

phonological rules which can be represented as finite state devices. The lexical component lists items of lexicon, stems and morphemes, in their base form and encodes constraints for the morphological alternation. The phonological rules are the ones we are more interested in this work.

For example, we can use Koskenniemi's formalism to write a phonological rule $p \Rightarrow \_q$. This rule can be read as "it is not the case that something ending in p is not followed by something starting with q" and "p is followed by q" [KaB05]. The 1x1 models are too simple to capture this kind of phenomenon, since they give us only one-to-one sound correspondences. Our more complex models, the 2x2 model and context model, are already able to capture these rules. When comparing the goodness of the models, we can use as a evaluation criteria the ability of a model to discover rules and also the complexity of those rules.

In principle, 2x2 model is able to discover all one-to-one, one-to-two, two-to-one and two-to-two rules. An example of two-to-two rule which cannot be described by the 1x1 model is a diphthong rule between Finnish and Estonian. According to it, Finnish diphthongs *uo*, *yö* and *ie* correspond to Estonian long vowels *oo*, *öö* and *ee*. Examples of typical rules found by the 2x2 model are shown in table 8.

The context model can discover even more complex rules which the 2x2 model can not describe. An example of this kind of rule is a phenomenon (also seen in the

Rule type	Finnish	Estonian	Description
1-to-1	*a*	*a*	A simple correspondence
1-to-2	.	*as*	Word ending *as* is not typical in Finnish
2-to-1	*a$*	*$*	At the end of a word Finnish *a* gets deleted
2-to-2	*uo*	*oo*	A diphthong is converted to a long vowel

Table 8: Examples of frequent rules of correspondence found by 2x2 model from alignment of Finnish and Estonian.

matrix of figure 11) where Finnish *t* corresponds to both Estonian *t* and *d*, Finnish *k* to Estonian *k* and *g* as well as Finnish *p* to Estonian *p* and *b*. Consonants *t*, *k* and *p* are voiceless, whereas *d*, *g* and *b* are voiced, all these consonants are plosives. This rule can be expressed in Koskenniemi's terminology in the following way:

> **Voicing of word-medial plosives in Estonian**: Voiceless plosive may change to voiced in Estonian, if not word-initial.

The rules found by context model are stored in the structure of the decision trees of the features. Thus, in order to find a rule that describes how the voiced feature behaves, we have to study the structure of the `VOICED` feature trees. The context model learns this rule and it can be found from a tree that is a `TARGET` level tree for `VOICED` consonant feature. The tree that contains this information is illustrated in figure 14. For clarity, in this example Finnish is the source level and Estonian the target level language, hence the the tree in the figure is for Estonian language.

The `VOICED` feature has two possible values, voiced and voiceless, which are denoted with + and −, respectively. In the count matrix of the root node (a.) in figure 14 is shown the distribution of voiced and voiceless consonants in the Estonian data. These values are distributed quite equally, there are slightly more voiceless than voiced consonants (821 vs. 801).

The inner nodes are circled with green color (nodes a., b., f. and j.) and the leaf nodes with blue (e. and g.) or with red (c., d., h. and i.). Note that the children of the node (j.) are not shown in this figure, since they are irrelevant in the task of finding the voicing rule. Each inner node of the tree is split according to the the best candidate to reduce the entropy of the count matrix, as described in section 4.7. In the root node the best splitting candidate is `(SOURCE, ITSELF, VOICED)` and it can be read as: "What is the value of the `VOICED` feature in the source language at this position?" There are three possible answers to this question, since the Finnish

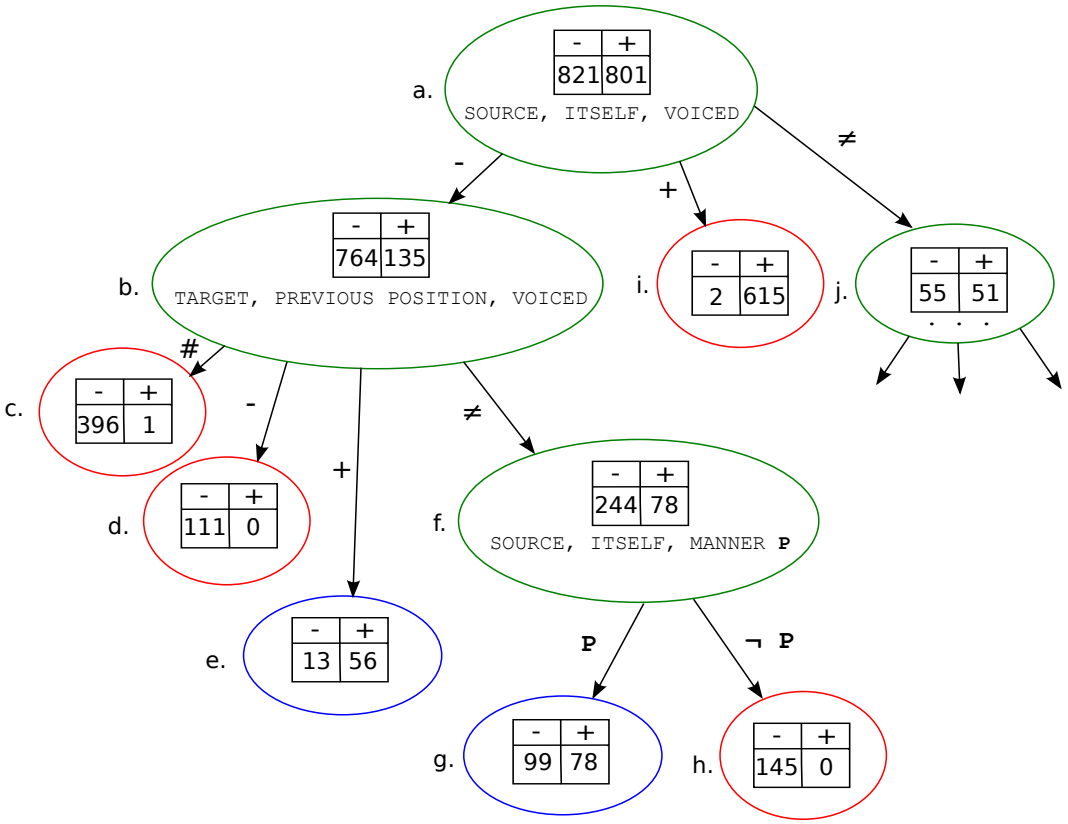

Figure 14: A context aware tree that shows the rule for medial voicing of plosives in Estonian.

symbol in the same position may be:

- a voiceless consonant$(-)$,
- a voiced consonant$(+)$
- or a symbol of wrong type $(\neq)$, i.e. a vowel or a dot.

The blue color reflects high entropy in the count matrix, red color low entropy. An example of zero entropy matrix is in (d.) node. In this node we can be sure that the Estonian consonant we are coding is voiceless. The path to this node gives us a rule: ”If the Finnish consonant at this position is voiceless *and* the previous Estonian symbol was a voiceless consonant, this symbol is certainly going to be a voiceless consonant too“.

The nature of the rule of voicing of word-medial plosives indicates that we will not find it from low entropy nodes. We have two possibilities to find this rule, the nodes

(e.) and (g.), where it may be so that the Estonian consonant is voiced. The rules that lead to these nodes say that voicing may occur if one of the following paths is satisfied:

Node e. The Finnish consonant at this position is voiceless *and* the previous Estonian symbol was a voiced consonant.

Node g. The Finnish consonant at this position is voiceless *and* the previous Estonian symbol was not a consonant (or the beginning of a word) *and* the Finnish consonant at this position was a plosive `P`.

Table 9 gives for each leaf node of the tree of figure 14 an example word pair that fulfills the conditions of the corresponding paths. For instance, a word pair for the node (d.) in the table is *koppa - kopp*. While coding the last Estonian consonant (which is *p* in this case), the fact that it is voiceless is determined by its context in the alignment: the Finnish symbol at the same position on the source level is voiceless and the second last Estonian symbol is also voiceless. In order to determine other feature values, we naturally have to discover all the target level trees.

Node	Word pair	The `VOICED` value FIN:EST
c	# k a . s a # **k** a d s a	voiceless:voiceless
d	# k o p p a # k o p **p** .	voiceless:voiceless
e	# s a m p i # s a m **b** .	voiceless:voiced
g	# i k ä # i **g** a	voiceless:voiced
h	# a s i a # a **s** i .	voiceless:voiceless
i	# a r v o # a **r** . u	voiced:voiced

Table 9: Examples of the voicing behavior of some Estonian consonants in the aligned word pairs. There is always exactly one leaf node in which the corresponding rule holds. The visualization of this `TARGET` level `VOICED` tree is in figure 14. Estonian consonant that is in focus is bold, the symbols that the rule utilizes are underlined.

The previous examples give an idea about the nature of the complex phonological rules that can be captured by the context model. This far we have studied rules of correspondences discovered by the context model between Finnish and Estonian, Finnish and Hungarian and Komi and Udmurt. The fact that the interpretation of the rules is very manual work, slows down the evaluation of them.

One problem we have faced with the rules learned by context model is the existence of "empty rules" that have learned trivial things about the sounds themselves. These rules do not give any new information for us but make the trees more complicated and harm the study of them. An example is in figure 15. The tree a is a `TARGET` level tree for the `SECONDARY` feature from Finnish-Hungarian alignment. The possible values of the `SECONDARY` feature are: none `n`, palatalized `'`, labialized `w` and aspirated `h`. The split is in the root node is conditioned with a query: "Is the value of the feature `PLACE OF ARTICULATION` dental `d` in this consonant?" Since all Hungarian consonants that are palatalized are also dental, we do not gain much from this rule. In any case this rule brings down the entropy as well as the code length. A question that needs to be answered is that if we could eliminate these rules, would they be replaced by more useful rules that still reduce the entropy.

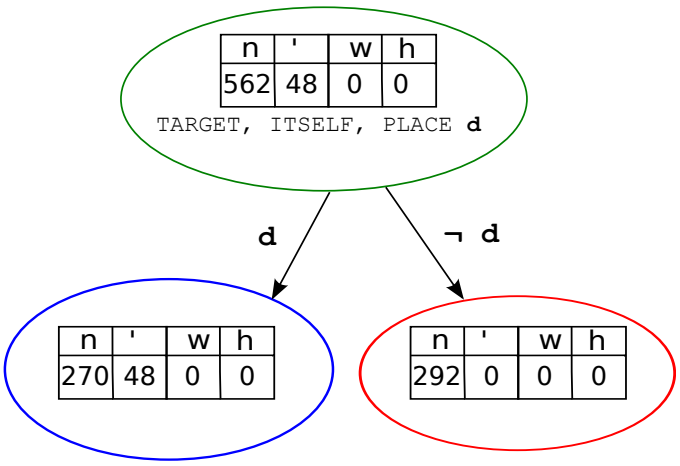

Figure 15: An empty rule in `TARGET` level tree for the `SECONDARY` feature in Finnish-Hungarian alignment.

5.5 Imputation

Imputation in statistics is a practice of substitution of values of missing data points. It helps to solve the missing-data problem. We use the term imputation in a slightly different purpose. The idea is to learn a model and predict (impute) unseen data with the help of the model. For instance, we could learn a model for aligning Estonian with Mordvin, by leaving out the word pair we are going to use in imputation. After the learning is finished, the model is given the unseen Estonian word, and the task of the model is to guess the corresponding Mordvin cognate. This procedure can be repeated for the full data set.

In the evaluation phase, we can compute the distance of the imputed words from the expected ones, and compare the performance of the models this way. Imputation is again an indirect way to evaluate the models, but it is objective and gives direct feedback about the strengths and the weaknesses of different models. The general imputation procedure consists of three steps:

1. The first step is the generation of the imputed word. For a given source word $\mathbf{s} = s_1 \dots s_n$ where $s_i \in S$, the task is to impute a target word $\mathbf{t} = t_1 \dots t_n$ where $t_i \in T_.$. The imputed word is the one that aligns with the source word with the lowest cost given the model used. The implementation of this step varies from model to model. The baseline imputation is described in section 5.5.1, 2x2 imputation in section 5.5.2, 3-dimensional imputation in section 5.5.3 and the context imputation in section 5.5.4.

2. The next task is to evaluate the goodness of the imputed word. We compute edit distance between the imputed and expected word. The edit distance can be computed on symbol or feature level.

3. Finally the imputation edit distance can be normalized and averaged over all words and we can compare the scores of different language pairs and also the scores between models.

5.5.1 Imputation for 1x1 Models

The imputation algorithm for the models that perform only 1x1 symbol alignments is very simple. Since these models know nothing about the context, each symbol s_i of the source word $\mathbf{s} = s_1 \dots s_n$ can be processed separately. Given a source symbol

$s_i \in S$, the imputed target symbol $t_j \in T_.$ is determined by the event cost in the model learned as shown in equation 22:

$$t_j = \arg\min_{t_j} L(s_i : t_j) \tag{22}$$

The event cost is computed as in equation 11. In principle, the 1x1 imputation is simply a mapping from source alphabet to the target alphabet.

The imputation of 1x1 symbol alignments can be illustrated with a look-up-table, as shown in figure 16. In this example the task is to impute a word in Estonian target alphabet $T_.$ for a Finnish word *kylmä*. Starting from the first symbol in the source word (k), its event count with each Estonian symbol in the global count matrix is checked. As seen from the global count matrix of figure 11, the Finnish k is most frequently aligned with target symbol $k \in T_.$. The task is repeated until we have found the lowest cost alignment for all the symbols in source word. According to the 1x1 symbol imputation algorithm, a deletion of source symbols is possible, but the insertion of target symbols is not. The resulting imputed word in Estonian alphabet is *külmä*, whereas the real Estonian cognate is *külm*.

	.	a	b	d	e	g	h	i	j	k	l	m	n	o	p	r	s	t	u	v	z	ä	õ	ö	ü
k										x															
y																									x
l											x														
m												x													
ä																						x			

Figure 16: Imputing Estonian from Finnish with a help of the 1x1 models.

5.5.2 Imputation for 2x2 Model

The imputation using 2x2 model is a bit more complicated, since the task is to find lowest cost mappings to target alphabet for all substrings of length 1 or 2 in the source word and find the lowest cost path through them. The solution to this problem is a dynamic programming approach.

The algorithm can be illustrated using the dynamic programming matrix as shown in figure 17. The example is designed for imputing Estonian counterpart for a Finnish

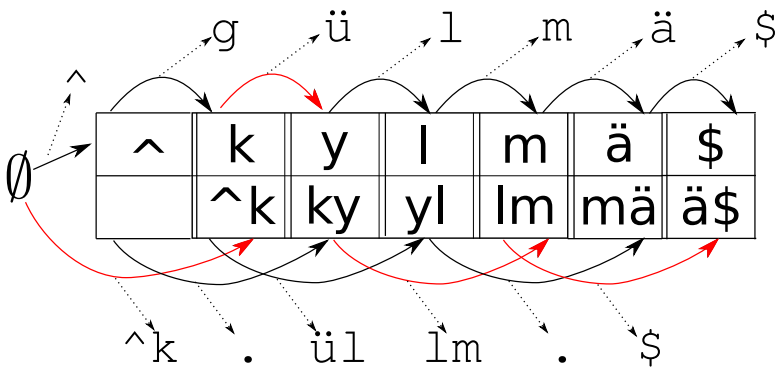

Figure 17: Dynamic programming matrix for imputing Estonian counterpart for a Finnish word *kylmä*.

word *kylmä*, given the learned 2x2 model. Each column i in the matrix stores the optimal path up to symbol s_i of the source word, and to each column leads 2 paths (except to column 1 is only possible to arrive from the beginning of the word). In order to end up to column i, the options are to read off a single source symbol s_i or a symbol sequence $s_{i-1}s_i$. The possible symbol alignments are: $(s_i : .)$, $(s_i : t)$, $(s_i : tt')$, $(s_{i-1}s_i : .)$, $(s_{i-1}s_i : t)$ or $(s_{i-1}s_i : tt')$, where t and t' are symbols of the target alphabet. The symbol alignment that minimizes the event cost is chosen, similarly to the equation 22, and attached to the lowest cost path.

After the path to column i is found, it is possible continue to the next column and repeat the procedure until the complete word is read off. The lowest cost path at the end of the word links to the resulting imputed word.

For instance, if we look at the symbol $s_2 = k$ in the source word, there are two possibilities to arrive to this column: from the beginning of the word or from column 1. The complete path in the first case is alignment sequence $(\hat{} : \hat{}) + (k : g)$ and the alternative path is to arrive directly, by aligning $(\hat{}k : \hat{}k)$. For this symbol, the cost of the latter path is lower. The complete path through the source word is drawn using red arrows, and it consists of alignments $(\hat{}k : \hat{}k)$, $(y : \ddot{u})$, $(lm : lm)$ and $(\ddot{a}\$: \$)$. The imputed word in Estonian alphabet is then *külm*, which happens to be exactly the same as the real Estonian cognate.

5.5.3 3-Dimensional Imputation

We use the 3-dimensional model to align cognates from languages S, T and V. The model can be used to impute words in third language, V, either with help of both words from S and T or just with one of them. For the incomplete examples (only one cognate from S or T is known), the imputation algorithm utilizes the corresponding marginal count matrix, G_{SV} or G_{TV} and is equal to the 2-dimensional case, as described in section 5.5.1.

For the complete examples, the imputation is done using dynamic programming. The 3-dimensional model learns only 1-to-1 sound correspondences, but the approach used with 1x1 models cannot be directly utilized, since the alignment of cognates **s** and **t** from languages S and T is not known. The 3-dimensional imputation algorithm *simultaneously* aligns the known word pair and imputes the third language. We follow the dynamic programming approach used in the 1x1 model building, as described in section 4.3.1.

The imputation is included in each step in which the cost of adding a new event into the best path that far is computed. For 2-dimensional models, the cost of adding an event was computed as in equation 8. In 3-dimensional imputation, the best path is searched only in 2 dimensions, but the event cost is computed in 3 dimensions. Equation 23 gives the cost of path in cell (s_i, t_j) of the dynamic programming matrix. The imputed symbol v_k is chosen to minimize the event cost L_{3D}.

$$C(s_i, t_j) = \min \begin{cases} C(s_i, t_{j-1}) & + \min_k L_{3D}(. : t_j : v_k) \\ C(s_{i-1}, t_j) & + \min_k L_{3D}(s_i : . : v_k) \\ C(s_{i-1}, t_{j-1}) & + \min_k L_{3D}(s_i : t_j : v_k) \end{cases} \tag{23}$$

The imputed word for language V is attached to the optimal alignment of **s** and **t**. It can be backtracked from the last cell of the matrix, by following the lowest cost path to the initial cell.

5.5.4 Context Imputation

The context imputation algorithm is using a context model to predict a target word $\mathbf{t} = t_1 \dots t_n$ where $t_i \in T$, given a source word $\mathbf{s} = s_1 \dots s_n$ where $s_i \in S$. Differently to the previously introduced imputation methods, a straightforward dynamic programming method cannot be applied, since the cost of an event $(s : t)$

cannot be computed independently from previously observed events, it depends on the context in which the event occurs. As a consequence, the lowest cost path of alignments up to symbol s_i in source word is possibly not optimal after seeing symbol s_{i+1}. In principle, all the paths should be carried along until all the symbols of $\mathbf{s}$ are aligned.

The context imputation algorithm is a greedy search heuristics that limits the number of paths to be considered in the mission of finding the sequence of target symbols $\mathbf{t}$ that align with the source word $\mathbf{s}$ with the lowest cost, using the previously learned context model $\mathcal{M}$. The complete context imputation algorithm is described in algorithm 2.

The context imputation algorithm tries to imitate the process of model learning. Thus, we have to allow deletions and insertions on both levels. We also have to set some restrictions to simplify the task. On source side, we allow maximum 3 dots in a row. Similarly to the restrictions in model building, it is forbidden to align a dot with a dot and a word boundary symbol can only be aligned with another word boundary symbol.

The fundamental idea of algorithm 2 is to read off source symbols one by one and at each positions i to collect B-best ways to impute the target symbols up to symbol s_i. The algorithm repeats a procedure: add the next source symbol s_i (or dot) and find the lowest cost target symbol $t \in T_{\cdot}$ to align it with. There are up to B different contexts, in which the symbol alignment $(s_i : t)$ can occur. The cost of each of these paths has to be computed separately.

The cost of adding a symbol alignment to a single path proceeds as follows. First, the source and target symbols are split to their feature representations, to determine the feature trees that are used to encode the event in the model. The contexts queried by the trees are given by the path of previous symbol alignments, so it is possible to find the leaf nodes, in which the the distribution of the feature values is stored. The cost of adding an event ϵ of type e, is given by equation 24, where $c(e)$ is the event count of e in the context model and $|E|$ is the number of all event types in the count matrix of the leaf node. The total cost of symbol alignment $(s_i : t)$ can be computed by summing up the event costs in all involved feature trees.

Algorithm 2 The Context Imputation

Input: Source word $\mathbf{s}$
Context model $\mathcal{M}$
Number of possible paths $\mathtt{B} = 100$
A list for saving the best alignments so far: $A_B = [(\hat{} : \hat{})]$
Output: Imputed word that consist of symbols from the target alphabet T

for position $j = 0 \rightarrow |\mathbf{s}|$ **do**
  Find the next source symbol s_j
  for all parent alignments $(\mathbf{s}_p : \mathbf{t}_p) \in A_B$ **do** {add 1st dot}
    Add $(\cdot : t)$ to the parent alignment and compute the total cost of the concatenated alignment for all $t \in T$
  end for
  Save $\mathtt{B}$ best alignments to a list A_{d1}
  for all parent alignments $(\mathbf{s}_p : \mathbf{t}_p) \in A_{d1}$ **do** {add 2nd dot}
    Add $(\cdot : t)$ to the parent alignment and compute the total cost of the concatenated alignment for all $t \in T$
  end for
  Save $\mathtt{B}$ best alignments to a list A_{d2}
  for all parent alignments $(\mathbf{s}_p : \mathbf{t}_p) \in A_{d2}$ **do** {add 3rd dot}
    Add $(\cdot : t)$ to the parent alignment and compute the total cost of the concatenated alignment for all $t \in T$
  end for
  Save $\mathtt{B}$ best alignments to a list A_{d3}
  {The next source symbol sequence $\mathbf{s}_j$ is one of the following:
  s_j, $\cdot s_j$, $\cdot\cdot s_j$ or $\cdot\cdot\cdot s_j$.}
  for all parent alignments $(\mathbf{s}_p : \mathbf{t}_p) \in A_B \cup A_{d1} \cup A_{d2} \cup A_{d3}$ **do** {add s_j}
    Add $(s_j : t)$ to the parent alignment and compute the total cost of the concatenated alignment for all $t \in T$.
  end for
  Clear A_B and save $\mathtt{B}$ best alignments to the list A_B
  position $j \leftarrow j + 1$
end for
Get lowest cost alignment from list A_B
return The imputed target word, dots removed.

$$
\begin{aligned}
P(\epsilon) &= \frac{c(e)+1}{\sum_{e'} c(e') + |E|} \\
L(\epsilon) &= -\log_2(P(\epsilon)).
\end{aligned} \tag{24}
$$

5.5.5 Edit Distance

We use *Levenshtein distance* to measure the difference between the imputed strings $\mathbf{t}'$ and expected strings $\mathbf{t}$. The metric, also called as *edit distance*, defines the minimum number of edits that are needed to transform one string into the other [Lev66]. The allowed edit operations are insertion, deletion and substitution, all of these also allowed by our models. Each edit operation has a cost, which is one if we want to compute the standard edit distance that gives the exact number of edit operations needed. We also define a *feature-wise edit distance*, where the feature representation of the symbols is used in case of a substitution operation. Then the cost is the proportion of identical features to the total number of features.

We repeat the imputation process for all words in source language that have a cognate in target language and obtain N examples. Then we compute the edit distance between each imputed word and correct word. If we sum the edit distances and normalize the sum by the size of the target language data, we get a measure called *normalized edit distance, NED*. If the edit distances are computed using feature-wise edit distance, this measure is called *normalized feature-wise edit distance, NFED*.

With the help of NED and NFED, we can compare the imputation power of different methods. In the scatter-plot of figure 18 is shown how the context model performs in comparison to a simple 1x1 two-part code model. There is shown the NFED of $10 * 9$ language pairs in both models, each symbol refers to one of the source languages. For most of the language pairs, the context model has a lower NFED. However, for some far-related languages the context model fails to find a better imputation result. This could be explained by the lack of data to lead the context model to learn stronger rules. The NED measure is never optimized directly by the models. However, there seems to be a correlation between the model cost and the normalized edit distance, which encourages to develop this evaluation method.

The normalized edit distance can also be used to measure the inter-language distances. Given a model, we can compute the NED or NFED for all language pairs in the data. The UPGMA-algorithm (as described in section 5.2) can be used to cluster the languages based on their edit distances. In figure 19 is a phylogenetic tree for Finno-Ugric languages, based on the imputation results from context models. If compared to figure 1, we can see how well the imputation approach can capture the language distances.

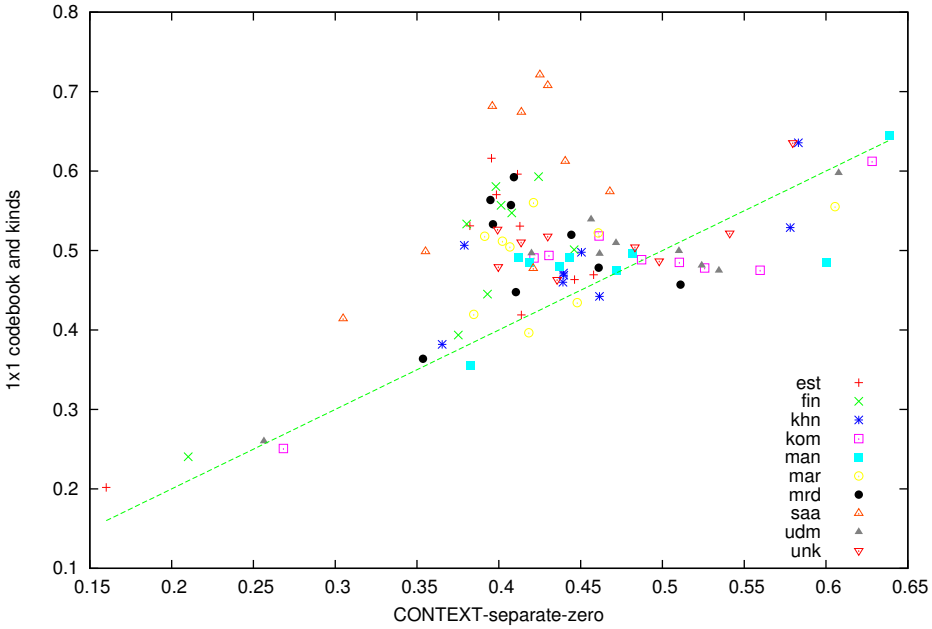

Figure 18: The comparison of imputation power of models in terms of normalized feature-wise edit distance. A context model on x-axis, 1x1 two-part code model on y-axis.

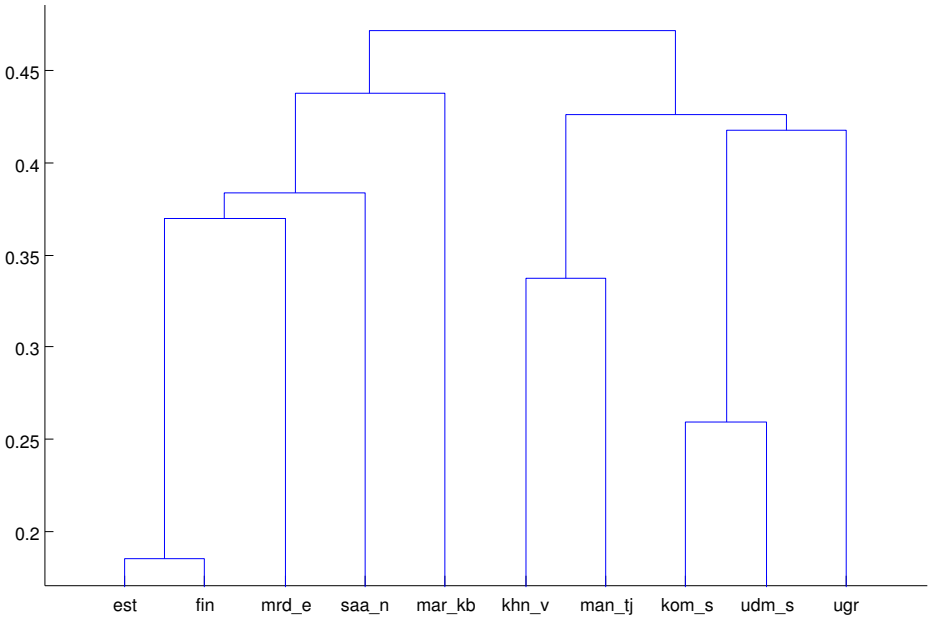

Figure 19: Finno-Ugric tree induced by normalized feature-wise edit distances in context model.

6 Conclusions

In this work, we have presented probabilistic models to induct alignment rules for etymological data. The data consists of cognate sets from Uralic language family, that is, sets of words that descend from the same origin. We used the data in its original form and in order to learn more about the sound change, we also defined a feature-based representation of sounds that is consistent across all the languages of the family.

All the models introduced in this work relied on the minimum description length principle. The approach is new in the computational etymology. We used prequential coding approach to encode the alignments of the words and optimized the models by minimizing the total code length of the data given the model in the spirit of two-part coding. We presented several models of varying complexity, the 1x1 models to capture simple 1-to-1 sound correspondences between two languages, 2x2 model for correspondences of up to 2 symbols and 3-dimensional model for aligning 3 languages simultaneously. The last model we described was a context model that learns phonetic rules that are conditioned by the environment of the sounds.

A challenging task was to design ways to test and evaluate the models. There is no straightforward way to implement the task, since there doesn't exist a gold standard in which to compare the alignments produced by our models. Since the basic idea of MDL is that learning can be viewed as data compression, it made sense to compare the compression power of the models to judge their goodness. The result was that our models perform very well in this specific task.

One of the main goals in the development of the models was to learn rules of regular sound correspondences. We noticed that even the simplest models can learn rules from the data and the context model captures many environmentally conditioned correspondences that are well-known in the etymological literature. We also computed the inter-language distances and normalized them to reconstruct the Uralic language tree, which was not identical to the tree drawn by linguistics, but preserved most of the relations.

The most ambitious evaluation schema was the imputation in which the models were used in a supervised way: after learning a model, it was used to predict unseen data. This approach gave promising results. We computed the normalized edit distances (NED) between the imputed and real cognates and used these scores to compare the models and language distances. There seemed to be a strong tendency that

a lower model cost indicates lower NED, which encourages to further develop the imputation methods. We also used the NED to induce a phylogenetic language tree which was already very close to the truth.

The goals of the future research aim both to improve the existing methods as well as to grow to new directions. The context model is currently only 2-dimensional and it could be extended to be three- or higher-dimensional. A related task is to develop a multi-dimensional context imputation algorithm. Multi-dimensional models would give us more information about the inter-language relations and would restrict the set of possible rules to be more accurate. The imputation methods can also be extended towards reconstruction of proto-forms of hypothetical parent languages which can be found from the internal nodes of the phylogenetic language tree. The reconstructed proto-forms would be very beneficial in the comparison of far-related languages and also give new information and support for the language evolution theories.

References

AdJ06 Adriaans, P. and Jacobs, C., Using mdl for grammar induction. In *Grammatical Inference: Algorithms and Applications*, Sakakibara, Y., Kobayashi, S., Sato, K., Nishino, T. and Tomita, E., editors, volume 4201 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2006, pages 293–306.

Ant89 Anttila, R., *Historical and comparative linguistics*. John Benjamins, New York, 1989.

BGK09 Bouchard-Côté, A., Griffiths, T. and Klein, D., Improved reconstruction of protolanguage word forms. *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009, pages 65–73.

BLG07 Bouchard-Côté, A., Liang, P., Griffiths, T. and Klein, D., A probabilistic approach to diachronic phonology. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, Prague, 2007, pages 887–896.

Bel57 Bellman, R., *Dynamic Programming*. Princeton University Press, 1957.

BeT93 Bertsimas, D. and Tsitsiklis, J., Simulated annealing. *Statistical Science*, (1993), pages 10–15.

BuW94 Burrows, M. and Wheeler, D., A block-sorting lossless data compression algorithm. Technical Report, Digital Equipment Corporation, 1994.

BWR09 Barbancon, F., Warnow, T., Ringe, D., Evans, S. and Nakhleh, L., An experimental study comparing linguistic phylogenetic reconstruction methods. *Proceedings of the Conference on Languages and Genes*, UC Santa Barbara, June 2009, Cambridge University Press, pages 887–896.

Čer85 Černỳ, V., Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45,1(1985), pages 41–51.

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

CrL02 Creutz, M. and Lagus, K., Unsupervised discovery of morphemes. *Proceedings of the ACL-02 workshop on Morphological and phonological learning - Volume 6*, MPL '02, 2002, pages 21–30.

Con07 Conery, J. S., Aligning sequences by minimum description length. *EURASIP Journal on Bioinformatics and Systems Biology*, 2007(2007), pages 4:1–4:14.

Cor01 Cormen, T., *Introduction to algorithms.* The MIT press, 2001.

Cov96 Covington, M. A., An algorithm to align words for historical comparison. *Computational Linguistics*, 22(1996), pages 481–496.

Cov98 Covington, M., Alignment of multiple languages for historical comparison. *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1.* Association for Computational Linguistics, 1998, pages 275–279.

CoT91 Cover, T. and Thomas, J., *Elements of information theory.* John Wiley & Sons, New York, NY, 1991.

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

CiV11 Cilibrasi, R. L. and Vitanyi, P. M., A fast quartet tree heuristic for hierarchical clustering. *Pattern Recognition*, 44,3(2011), pages 662 – 677.

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.

GMP05 Grünwald, P., Myung, I. and Pitt, M., *Advances in minimum description length: Theory and applications.* The MIT Press, 2005.

Gol01 Goldsmith, J., Unsupervised learning of the morphology of a natural language. *Computational linguistics*, 27,2(2001), pages 153–198.

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

Ham86 Hamming, R., *Coding and information theory*. Prentice-Hall, Inc., 1986.

Huf52 Huffman, D., A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40,9(1952), pages 1098–1101.

Itk00 Itkonen, E. and Kulonen, U.-M., *Suomen Sanojen Alkuperä (Origin of Finnish Words)*. Suomalaisen Kirjallisuuden Seura, Helsinki, Finland, 2000.

Juo98 Juola, P., Measuring linguistic complexity: The morphological tier. *Journal of Quantitative Linguistics*, 5,3(1998), pages 206–213.

KaB05 Karttunen, L. and Beesley, K., Twenty-five years of finite-state morphology. *Inquiries into Words, Constraints and Contexts*, (2005), page 71.

KGV83 Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P., Optimization by simulated annealing. *Science*, 220,4598(1983), pages 671–680.

KMT96 Kontkanen, P., Myllymäki, P. and Tirri, H., Constructing Bayesian finite mixture models by the EM algorithm. Technical Report NC-TR-97-003, ESPRIT Working Group on Neural and Computational Learning (NeuroCOLT), 1996.

Kol65 Kolmogorov, A., Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1,1(1965), pages 1–7.

Kon02 Kondrak, G., Determining recurrent sound correspondences by inducing translation models. *Proceedings of COLING 2002: 19th International Conference on Computational Linguistics*, Taipei, August 2002, pages 488–494.

Kon03 Kondrak, G., Identifying complex sound correspondences in bilingual wordlists. *Computational Linguistics and Intelligent Text Processing (CICLing-2003)*, Gelbukh, A., editor, Mexico City, February 2003, Springer-Verlag Lecture Notes in Computer Science, No. 2588, pages 432–443.

Kon04 Kondrak, G., Combining evidence in cognate identification. *Proceedings of the Seventeenth Canadian Conference on Artificial Intelligence (Canadian AI 2004)*, London, ON, May 2004, Lecture Notes in Computer Science 3060, Springer-Verlag, pages 44–59.

Kon09 Kondrak, G., Identification of cognates and recurrent sound correspondences in word lists. *Traitement automatique des langues*, 50,2(2009), pages 201–235.

Kos83 Koskenniemi, K., *Two-level morphology: A general computational model for word-form recognition and production.* Ph.D. thesis, University of Helsinki, Finland, 1983.

KoS06 Kondrak, G. and Sherif, T., Evaluation of several phonetic similarity algorithms on the task of cognate identification. *Proceedings of the Workshop on Linguistic Distances*, LD '06, Stroudsburg, PA, USA, 2006, Association for Computational Linguistics, pages 43–50.

KSL06 Kettunen, K., Sadeniemi, M., Lindh-Knuutila, T. and Honkela, T., Analysis of eu languages through text compression. *Advances in Natural Language Processing*, (2006), pages 99–109.

Lev66 Levenshtein, V., Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, volume 10, 1966, pages 707–710.

LiV94 Li, M., and Vitányi, P., *An Introduction to Kolmogorov Complexity and Its Applications.* Springer-Verlag New York Inc, 1994.

Lyt73 Lytkin, V. I., *Voprosy Finno-Ugorskogo Jazykoznanija (Issues in Finno-Ugric Linguistics)*, volume 1–3. Nauka, Moscow, 1973.

Mel97 Melamed, I., Automatic discovery of non-compositional compounds in parallel data. *2nd Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Providence, RI, June 1997.

Mel00 Melamed, I., Models of translational equivalence among words. *Computational Linguistics*, 26,2(2000), pages 221–249.

Mur84 Murtagh, F., Complexities of hierarchic clustering algorithms: the state of the art. *Computational Statistics Quarterly*, 1,1(1984), pages 101–113.

NRW05 Nakhleh, L., Ringe, D. and Warnow, T., Perfect phylogenetic networks: A new methodology for reconstructing the evolutionary history of natural languages. *Language (Journal of the Linguistic Society of America)*, 81,2(2005), pages 382–420.

NeW70 Needleman, S. B. and Wunsch, C. D., A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48,3(1970), pages 443 – 453.

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.

PWN09 Prokić, J., Wieling, M. and Nerbonne, J., Multiple sequence alignments in linguistics. *Proceedings of the EACL 2009 Workshop on Language Technology and Resources for Cultural Heritage, Social Sciences, Humanities, and Education*. Association for Computational Linguistics, 2009, pages 18–25.

QuR89 Quinlan, J. and Rivest, R., Inferring decision trees using the minimum description length principle. *Information and Computation*, 80,3(1989), pages 227–248.

Réd91 Rédei, K., *Uralisches etymologisches Wörterbuch*. Harrassowitz, Wiesbaden, 1988–1991. URL `www.kotus.fi/julkaisut/sanakirjat/ssa/ssa.html`.

RHM06 Roos, T., Heikkilä, T. and Myllymäki, P., A compression-based method for stemmatic analysis. *Proceeding of the 2006 conference on ECAI 2006: 17th European Conference on Artificial Intelligence August 29– September 1, 2006, Riva del Garda, Italy*. IOS Press, 2006, pages 805–806.

Ris78 Rissanen, J., Modeling by shortest data description. *Automatica*, 14,5(1978), pages 465–471.

RaK09 Ravi, S. and Knight, K., Minimized models for unsupervised part-of-speech tagging. *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, 2009, pages 504–512.

RuN10 Russell, S. and Norvig, P., *Artificial intelligence: a modern approach.* Prentice Hall, 2010.

RWT02 Ringe, D., Warnow, T. and Taylor, A., Indo-european and computational cladistics. *Transactions of the Philological Society*, 100,1(2002), pages 59–129.

Sha48 Shannon, C., A mathematical theory of communication. *The Bell System Technical Journal*, 27,7(1948), pages 379–423.

Sin97 Sinor, D., editor, *The Uralic Languages: Description, History and Foreign Influences (Handbook of Uralic Studies Vol 1)*. Brill Academic Publishers, Wiesbaden, August 1997. URL `www.kotus.fi/julkaisut/sanakirjat/ssa/ssa.html`.

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

Sta05 Starostin, S. A., Tower of babel: Etymological databases, 2005. URL `http://newstar.rinet.ru/`.

WHY10 Wettig, H., Hiltunen, S. and Yangarber, R., Hidden markov models for induction of morphological structure of natural language. *Proceedings of WITMSE-2010: Workshop on Information Theoretic Methods in Science and Engineering*, Tampere, Finland, August 2010.

WHY11a Wettig, H., Hiltunen, S. and Yangarber, R., Mdl-based modeling of etymological sound change in the uralic language family. *Proceedings of WITMSE-2011: The Fourth Workshop on Information Theoretic Methods in Science and Engineering*, Helsinki, Finland, 2011.

WHY11b Wettig, H., Hiltunen, S. and Yangarber, R., MDL-based models for alignment of etymological data. *International Conference on Recent Advances in Natural Language Processing (RANLP 2011)*, Hissar, Bulgaria, September 2011, pages 111–117.

WaP93 Wallace, C. and Patrick, J., Coding decision trees. *Machine Learning*, 11(1993), pages 7–22.

WeY11 Wettig, H. and Yangarber, R., Probabilistic models for alignment of etymological data. *Proceedings of 18th Nordic Conference of Computational Linguistics NODALIDA 2011*, Riga, Latvia, 2011.

ZiL77 Ziv, J. and Lempel, A., A universal algorithm for sequential data compression. *Information Theory, IEEE Transactions on*, 23,3(1977), pages 337–343.