European master in Language and Communication Technologies

AUTOMATED MULTILINGUAL COGNATE RECOGNITION & TRANSITION-RULE EXTRACTION

Master's thesis.

By:

M·S·KROON, BA

Supervisors:

dr. G·BOUMA[†]
dr. É·BUCHI[‡]

RIJKSUNIVERSITEIT GRONINGEN[†]
&
UNIVERSITÉ DE LORRAINE[‡]

XXIV.VIII.MMXVI

Abstract

The task of automated cognate recognition is useful for many fields of linguistics, but only a handful of studies have dealt with multiple languages at once. In the research put forth in this thesis a system is developed and evaluated that can automatically recognize cognates throughout multiple languages using multilateral transition rules. An SVM is also tested on the same data. It was found that adding more languages in the equation of cognate recognition and using multilateral transition rules improves cognate recognition. The result of this thesis is a list of extracted cognate tuples, a list of multilateral transition rules and their probability, and a multilingual cognate-recognition system.

Acknowledgements

First and foremost I would like to thank my supervisors, Gosse Bouma and Eva Buchi, for their invaluable help, their thorough reading and hard (sometimes late-night) work.

I would also like to thank Maxime Amblard, Miguel Couceiro and Gosse (this time as the local coordinator of the LCT programme in Groningen) for making sure that everything went as smoothly as possible with respect to the administrative differences between the two universities.

Finally, I would like to thank my parents, my brother, my friends and Masha for feeding me, their support and their love, without which this thesis would not have been possible.

Contents

1. Introduction

For centuries linguists have researched the genetic relationship between languages. Languages have been classified into language families, where each member of the family was derived from a common ancestral. Even within these language families, languages are classified into branches of languages that exhibit even more lexical and grammatical similarities. English, for example, is a member of the Indo-European language family, and more specifically a member of the Germanic branch of the Indo-European languages. More specifically yet, English is classified as a West-Germanic language, just as Frisian, Dutch, Afrikaans and German are, among many other languages (Lewis, Simons, & Fennig, 2016). These West-Germanic languages share more linguistic phenomena than languages from different branches, such as English and Nepali, an Indo-Arian (and thus an Indo-European) language – these languages are eventually related, but the two languages diverged much earlier than the languages in the West-Germanic branch did, and have therefore undergone many different changes over time and therefore bear fewer resemblances.

Languages can, for example, undergo certain phonological changes. If a phonological change occurred in one group of languages, but did not occur in another group, that means that the two groups diverged before the change took place and that the languages inside the groups diverged later. For example, Grimm's law (McColl Millar & Trask, 2007) describes a set of sound changes that only occurred in the development of Proto-Indo-European (the common ancestral of all Indo-European languages) into Proto-Germanic (the common ancestral of all Germanic languages); Grimm's law did not occur in any other branches of the Indo-European languages.

Words between two languages that have the same etymological origin are called cognates. Sometimes the cognatic relation between two words is not as obvious, as for example in the French *lait* 'milk' and the Ancient Greek *γάλα* (gen. *γάλακτος*) 'milk', but the more related the languages are (i.e. the later the languages diverged and therefore the more sound changes they have undergone together), the easier it is to recognize two cognates. In the Germanic languages (both West- and North-Germanic), recognizing cognates is relatively easy. Usually, cognate recognition is done by the use of string resemblance: cognates tend to share graphemic features. One could (and some do) also use sound correspondences, such as 'if Dutch has <d>, then English has <d>', to recognize cognates. See chapter 2 for a more detailed explanation of different approaches.

However, sound correspondences between two languages may not be very exact. For instance, the example above is not true in every case: in many cases it is the case that 'if Dutch has <d>, then English has <th>'. The probability that Dutch <d> actually corresponds to English <d> is therefore not very high, as it often corresponds with English <th>. The correspondences <d>:<d> and <d>:<th> are not the most reliant. If another language is added to the equation, though, the correspondences get a much higher confidence (or probability), such as 'if Dutch has <d> and German has <t>, then English has <d>' and 'if Dutch has <d> and German has <d>, then English has <th>'. These correspondences (or rules) are true in almost a 100% of the cases. Adding another language in the equation of cognate recognition should thus give better results: the calculated probability for a triple of words being cognates must be more accurate than the probability for a pair of words.

Algorithms and natural language processing (NLP) tools have been developed for automated cognate recognition (Kondrak, 2001; Bergsma & Kondrak, 2007a; Cysouw & Jung, 2007; List, 2012;

Wang & Sitbon, 2014; Rama, 2015; Xu, Chen, & Li, 2015, among others). All of these systems, however, only take two languages as input, while the results should be better in terms of accuracy, precision and recall if it considers multiple languages – only a handful of studies have dealt with multiple languages at once, but these systems do not produce correspondence rules (Bergsma & Kondrak, 2007b; Hall & Klein, 2010, 2011, among others). I hypothesize, however, that cognate recognition can be improved by using sound-correspondence rules between multiple languages to come to a more confident decision (i.e. the set of cognates will have a higher probability of being cognates). The research put forth in this thesis, therefore, focuses on the development of a system for automated cognate recognition that takes more than two languages as input and which returns a list of observed sound-correspondence rules and their probabilities, and can calculate the probability that all words in a tuple are cognates.

The system was developed based on Germanic languages only – in particular on three West-Germanic (English, German and Dutch) and two North-Germanic languages (Danish and Swedish). The system was also evaluated on these languages. The result of this thesis is a universal[1] multilingual automated cognate recognition system, a list of extracted tuples of Germanic cognates using said system, and a set of discovered sound-correspondence rules. Excerpts of the latter two can be found in appendix A and B respectively.

In chapter 2, I shall place this research in the existing literature. In chapter 3, I shall explain the outlay of this research. Results of the system can be found in chapter 7, a discussion of the thesis can be found in chapter 8.

Before everything else, it needs to be discussed how some technical terms are used in this thesis.

Cognate: Trask (2000) defines a *cognate* as "[a] language or a linguistic form which is historically derived from the same source as another language/form". In this thesis, *cognates* are confined to linguistic forms that are derived from the same source; the possible meaning of *cognate language* is not used here. In this research the term *cognate* also excludes doublets. Loans are considered cognates. Calques[2] of which all morphemic constituents are cognates with the corresponding morphemic constituents from the source are also considered cognates. E.g. the Swedish word *skyskrapa* is a calque of the English *skyscraper*: since both parts, *sky* and *skrapa*, are cognates of English *sky* and *scraper* respectively, the word pair *skyscraper*-*skyskrapa* is considered a cognate pair. Furthermore, I do not consider semantics for *cognates*; words can be cognates, even though meaning something else (false friends).

Cognacy: *Cognacy* is used as the state of being cognates.

Etymon: Trask (2000) defines an *etymon* as "the form from which a later form derives". I use this in the same manner.

Transition: In this thesis I use the word *transition* as meaning *correspondence*. Even though *correspondence* is the term used in comparative linguistics, I decided to use *transition* because the cognate recognition system developed can to some degree be considered a weighted finite-state transducer, where the term *transition* is more commonly used. Hence also the notation `a:b/`p (as these notations are used in weighted finite-state transducers as well), meaning that `a` and `b` transition into each other (or correspond) with a probability p.

[1]The system was written in such a way that it can process any number of any languages: it does not only work on five Germanic languages.

[2]Trask (2000) defines a calque as "a type of borrowing, where the morphemic consituents of the borrowed word or phrase are translated item by item into equivalent morphemes in the new language".

2. Background

The task of automated cognate recognition is useful for many fields of linguistics. It has of course been a central problem in historical and comparative linguistics to detect cognates, and with the increasing amount of data in historical linguistics, the need for automated methods for cognate recognition grew as well. Cognates are also used for measuring language change and language diversity; pairs of cognate words can be used as indicators of the perceived difficulty of texts for L2-learners. In machine translation cognates are useful, too, with cognate words not seldom being the best translation of each other (Kondrak, Marcu, & Knight, 2003).

Although automated cognate recognition seems to be a solved problem, the existing systems can be improved (humans can still recognize cognates better). As I hypothesize, using sound-correspondence rules between multiple languages to come to a more confident decision, which has never been done before, should benefit automated cognate recognition and improve results.

The majority of the automated cognate recognition systems that have been developed already use orthographical information only (Simard, Foster, & Isabelle, 1993; Danielsson & Muehlenbock, 2000; Mann & Yarowsky, 2001; Inkpen, Frunza, & Kondrak, 2005; Mulloni & Pekar, 2006; Bergsma & Kondrak, 2007b, among others), and are based on string resemblance between two possible cognates. For this, several different techniques have been investigated, such as the Levenshtein distance (Levenshtein, 1966), XDICE, (Brew, McKelvie, et al., 1996) and LCSR (Melamed, 1995). Some other systems take into account phonetic information of words as well to recognize cognates (Guy, 1994, among others). My system as put forth in this thesis will only focus on orthographical information of words, however its design allows for phonetic input as well: it will be able to work with texts that are transcribed in IPA (or any other phonetic notational system), but this will require redefining what consonant, vowel and semivowel characters are for the system first.

A handful of systems focus on distinguishing cognates that have the same meaning from false friends by implementing semantic information (i.e. meaning) of words (Kondrak, 2004; Wang & Sitbon, 2014). Brew, McKelvie, et al. (1996) and Frunza and Inkpen (2006) do this by using parallel corpora: the matched cognates have the same meaning and therefore are not false friends. Kondrak (2001) uses the words' dictionary definition to measure their semantic distance. The list of extracted cognates that is one of the results of this thesis will not contain false friends either, as the cognates are extracted form a parallel corpus (see chapter 3 and section 4.1). The developed system will, however, be able to recognize false friends as cognates – this would require another presentation of the data (i.e. a non-parallel corpus).

Malmasi, Dras, et al. (2015) combine the string-distance approach and the parallel-corpus approach by aligning a parallel sentence on a word level and detecting pairs of cognates based on the Jaro-Winkler distance (Winkler, 1990). This resulted in a system with a performance with an f-score of 0.63. My approach is rather similar but takes one step more. A parallel corpus is aligned on a word level, and all word tuples with a relatively low Jaro-Winkler distance are removed.[1] This results in a list of possible cognates. My system then calculates a probability of the words in the tuple actually being cognates; if the probability is below a certain threshold, they are not considered cognates (see chapters 3 and 6 for a description of the developed system).

[1]The Jaro-Winkler distance, confusingly, is more a similarity measure than a distance. Therefore, the lower the Jaro-Winkler distance, the less alike the words are.

My system calculates the probability that words in a tuple are cognates through a set of phoneme correspondences it has found earlier (or grapheme correspondences if the input is not phonetically transcribed; these correspondences I call transitions (see chapter 1)). Few other systems take into account transitions (Barker & Sutcliffe, 2000; Koehn & Knight, 2000; Gomes & Lopes, 2011, among others). Mulloni and Pekar (2006) did not consider transition rules, and noticed that many errors in cognate detection by their system were due to single-letter substitutions that are incorrect. They suggest that a possible solution to this problem could be to introduce weighting for detected transition rules: this is exactly what my system does. Guy (1994) did a similar thing to calculate probabilities for transitions using the chi-square statistic, but he only did this in words with the same meaning, thus ignoring false friends. My system does not make a distinction between same-meaning cognates and false friends, and detects transition rules between cognates in the broader sense, as defined by me in chapter 1.

So in short, this research places itself among the existing literature in the following way: i) my system recognises cognates throughout more than two languages, whereas most systems deal with language pairs; ii) though all existing systems that deal with more than two languages do not consider transition rules, my system does; iii) although using only orthographical information of the input tuples, my system does not recognize cognates based on string resemblance but rather on discovered transition rules, which the majority of the other systems do not; the system does have the ability to process phonetically transcribed data, but this would require slight tweaking; and iv) my system uses semantic information of possible cognates to determine if they are in fact cognates by extracting the tuples of possible cognates from a parallel corpus, comparably to some other existing systems.

3. Outline of the research

In this chapter it shall be explained how the problem of cognate recognition and the compilation of a cognate database will be tackled.

Cognate recognition will be performed on a list of possible cognates. It would be ill-considered to take all words from a corpus for a few languages and test all combinations of words for cognacy; this will result in a multitude of combinations (with five corpora of 1,000,000 words each, the number of combinations would already be 10^{30}; in reality, the used corpora contain up to 50.6 million words, resulting in an even larger number of combinations) which will simply take too much time, will likely result in memory overflows for a computer, and would be unnecessary. It would be unnecessary because it is clear as day that the words *Zimbabwean* and *tafeltenniscoach* are not cognates: the majority of combinations do not even have to be considered as possible cognates and can be ignored.

The list of possible cognates must therefore be confined. In order to do this, a parallel corpus will be used, which will be aligned on word level, so that words that are each other's translation are aligned; in closely related languages cognates are not seldom translations of each other. The number of possible cognate tuples shall be further diminished by removing those tuples in which the lengths of the words differ too much and which have too low a string resemblance. The data selection will be discussed in 4.1.

Before cognate recognition can take place, the computer needs to learn what cognates look like. For this purpose, I have written a machine learning algorithm that: i) discovers substring-transition rules for cognate tuples[1] of any length (such as 'if Dutch has a <d>, English has <th>' and 'if Dutch has a <d> and German has a <t>, English has <d>') and calculates their probabilities; ii) finds the best substring alignment for a tuple of possible cognates (such as in Figure 3.1); and iii) then calculates the probability that all words in the tuple are cognates of each other. It then calculates a probability threshold beyond which the words in the tuple are or are not cognates.

^ch	u	r	ch	$\epsilon$$
^K	i	r	ch	e$
^k	e	r	k	$\epsilon$$

Figure 3.1: An example of substring alignment for an English, German and Dutch word. 'ϵ' represents the empty string, '^' represents the beginning of a word, '$' represents the end of a word.

Albright and Hayes (2006) presented the Minimal Generalization Learner, a system that learns transition rules between word pairs and can apply these rules to a set of words to predict an expected form. Their system works rather well, for example, on discovering the paradigm of English past tenses, where it will discover in which cases past tenses are formed with the suffix -/d/, with the suffix -/ɪd/ or rather with a vowel change such as in strong verbs. In the system developed for this research, these kinds of transition rules need to be discovered as well. However, the MGL

[1] The technical term *cognate tuple* corresponds to what is called in historical linguistics a *cognate set*, which is defined as a set of words "which are directly descended from a single ancestral form in the single common ancestor of the languages in which the words [...] are found, with no borrowing" (Trask, 2000, p. 62).

only works with one-way transitions, and only for input pairs (such as present tense-past tense). I therefore had to develop a multilateral transition learner that can discover n-way transition rules between input tuples of any length.

Substring alignment has been researched before. Covington (1996) presents an algorithm to find the best substring alignment of two strings. However, their algorithm does not use weights for possible alignments other than preferring consonant-to-consonant transitions and vowel-to-vowel transitions. The alignment is also done with substrings being single characters only, whereas my system should be able to align substrings of different lengths. In grapheme-to-phoneme alignment (Pagel, Lenzo, & Black, 1998, among others), aligning substrings of different lengths is an issue as well, but grapheme-to-phoneme alignment deals with input pairs only. I therefore had to design a substring aligner that aligns any number of strings on a substring level, that can align substrings of different lengths (such as `th` with `d`) and that uses transition-specific weights to find the best alignment.

A discription of the developed algorithm can be found in chapter 6. The training, development and test data that were given to the system, will be discussed in section 4.2.

The system will also calculate the average of the probabilities it assigns to the tuples that are cognates (which can be found in the training, development and test set). Not only does it calculate the observed probability, but also the expected probability based on the probabilities of subsets of words in the tuple being cognates. For example, for the transition tuple (a, b, c) it will calculate the observed probability $P(a : b : c)$, as well as the expected probability based on $P(a : b)$, $P(a : c)$ and $P(b : c)$. How the observed and expected probabilities can be compared is shown in chapter 5.

This machine learning algorithm will be run on all combinations of the five languages on which this research focuses: English, Danish, German, Dutch and Swedish – the reason why these languages are focused on is because the corpus used for this research (Europarl (Tiedemann, 2012); see section 4.1) is available for those five Germanic languages; and Europarl is used because it is available for so many Germanic languages. All combinations are tested so that results can be compared, to see if results improve with more languages. A support vector machine (SVM) will be run on the data as well, to compare my system's performance. Ciobanu and Dinu (2014) experiment with SVMs to learn orthographic changes (i.e. transitions) and to detect cognate pairs, too.

For the cognate extraction, the system will be run on prepared tuples of possible cognates extracted from the Europarl corpus. It will then write all tuples for which it calculated a probability higher than the established threshold to a file.

4. Data

4.1 The Europarl corpus

The corpus on which the developed system was run is a part of the freely accessible Europarl corpus (Tiedemann, 2012), of which the English-Danish, English-German, English-Dutch, and English-Swedish parallel corpora were used. The Europarl corpus was used because of its parallel nature, which was very convenient for the approach that was used (for which see chapter 3). This led to the choice of using the Europarl corpus, along with Europarl being the parallel corpus that is available in the most Germanic languages.

The Europarl corpus is a collection of proceedings of the European Parliament starting from 1996. The latest update was released in 2012; it now contains sentence-aligned texts for 21 languages, containing 753.73 million tokens and 30.11 million sentences. For most languages lemmas, part-of-speech (POS) tags and sometimes even morphological tags are provided for each word in every sentence as well. For this research, only sentence-aligned texts for English and Danish, German, Dutch and Swedish were used, totalling to 372.9 million words and 7.75 million sentences. See table 4.1 for the number of sentences and words throughout the used parts.

	Sentences	English words	L2 words
English-Danish	1,968,800	48,574,988	44,654,417
English-German	1,920,209	47,818,827	44,548,491
English-Dutch	1,997,775	49,469,373	50,602,994
English-Swedish	1,862,234	45,703,795	41,508,712

Table 4.1: The number of sentences and words in the used parts of the Europarl corpus.

The Europarl corpus files were retrieved from the OPUS (open parallel corpus) website by Tiedemann (2012).[1] OPUS is an online collection of texts, which are provided with linguistic annotation. However, the compiled collection has not undergone manual corrections, meaning that the data can be somewhat noisy.

Along with the texts and their annotations, some linguistic tools have been made available on the website as well, such as tools for tagging and parsing. In order to prepare the data for this research, I used a tool that replaces all words in the Europarl data by their lemmas with POS tags. However, for Danish and Swedish lemma information is not provided. For Swedish, at least morphological tags are provided (i.e. very specific POS tags which also contain morphological information), which were later used for lemmatization.

The resulting files with the aligned texts were then joined into one large file, so that all sentences that did not occur throughout all five languages were removed. The result was a file with 1,401,234 lines with five sentences in five languages meaning the same thing, with words reduced to a lemma–POS-tag tuple where possible.

[1] http://opus.lingfil.uu.se/

4.1.1 Lemmatization

As for the languages for which no lemmas were provided, Danish and Swedish, the texts had to be lemmatized in order for the alignment on word level and the cognate recognition to be more accurate. Even though there are several lemmatizers for Danish and Swedish, there were no open-access lemmatizers. Jörg Tiedemann did suggest using Robert Östling's pipeline available from GitHub, but after long and hard trying I could not get it working. I was also recommended a lemmatizer for Danish, but it required database files I had no access to. I then decided to write simple lemmatizers for Danish and Swedish myself.

The Danish lemmatizer takes a relatively small word-to-lemma database (18,390 tuples) from the Danish Dependency Treebank v1.0 (DDT) (Kromann & Lynge, 2004). The lemmatizer takes this as its lexicon. It then takes a word from the corpus, along with its POS tag, and sees if it can find it in the lexicon taken from the DDT. If not, it makes a set of all possible stems from which the word could have been derived given the Danish morphology and the word's POS tag. If a stem does not contain any vowel, it is ignored. Then, if a stem has a common ending, that is to say a derivational-morphologically productive suffix, that stem will be taken as the lemma. Endings considered common are: *-ion, -ing, -else, -hed*, and *-itet* for nouns, *-sk* for adjectives and *-ere* for verbs. If the word does not have a common ending, the lemmatizer checks if any of the stems in the set of possible stems is in the lexicon. It also considers the possibility that the word is a compound, trying to split it in such a way so that all parts, minus possible linking morphemes, are in the lexicon (or have a common ending). Because of the way Germanic compounds work, only the POS tag of the last part of the potential compound has to correspond to the POS tag of the word; all other parts can be of any POS tag. If the system cannot find a suitable lemma for the word using any of these techniques, the word will be added to the lexicon, mapped to itself as lemma: the lemmatizer simply does not know the word in any way, and it would be very time consuming to walk through the whole algorithm checking for the word's lemma every time the system encounters it. For the pseudo-code, see Algorithm 1.

Algorithm 1: Danish lemmatizer

```
Result: Reduce a Danish word to its lemma
Read DDT lexicon;
for word do
    if word in lexicon then
        return lemma;
    else
        make set of possible stems;
        for possible stem do
            if pos. stem contains a vowel then
                if pos. stem has common ending then
                    return lemma;
                else if pos. stem in lexicon or compound then
                    return lemma;
        else
            save word to lexicon;
            return word
```

The part of the Danish lemmatizer that checks whether a word is a compound, first checks if the word is in the lexicon. If it is not, it breaks the word up into syllables, using the `pyphen` package for Python, and then gives back all possible partitions of those syllabifications. Then for all possible partitions, for all parts in the partition, it makes a set of all possible stems the part could have been derived from, stripping them from all possible linking morphemes, except for the last part. If, then, for every part at least one of those possible stems is in the lexicon or has a common ending (ignoring POS tags), and the last part is in the lexicon (taking into account its POS tag), the compound is added to the lexicon, and the algorithm returns the lemma by joining all non-final parts and the lemma of the last part. For pseudo-code, see Algorithm 2.

Algorithm 2: Compound checker

```
Result: Return the lemma of a compound
if word in lexicon then
    return lemma
else
    syllabify word;
    for possible partition of syllables do
        for part in pos. partition do
            make set of stems;
            if (any of those stems in lexicon or has common ending) and last part in lexicon
            then
                add word to lexicon;
                return non-final parts + last part's lemma
```

The Swedish corpus is provided with POS tags containing morphological information. The Swedish lemmatizer is therefore based on these tags, as every tag already implies the morphemes to be stripped and the lemma from which the word was derived. Nevertheless, to improve speed and accuracy, the Swedish lemmatizer uses a lexicon: SALDO (Borin, Forsberg, & Lönngren, 2013), an extensive Swedish lexicon containing semantic and morphological information. The lemmatizer was also given the most frequent (if not all) irregular inflections, retrieved from the Learning Swedish website (Swedish Institute et al., 2015).[2] For every word, the lemmatizer first checks if it is in the lexicon. If it is, the word is reduced to the lemma as provided in SALDO. If it is not, the lemmatizer analyses the morphological POS tag accompanying the word, and checks if the word is irregular or strips it of the appropriate morphemes, taking into account some spelling or phonological processes that may occur. For pseudo-code, see Algorithm 3.

[2]http://learningswedish.se/

Algorithm 3: Swedish lemmatizer

```
Result: Reduce a Swedish word to its lemma
Read SALDO lexicon;
Read irregulars from Learning Swedish;
for word do
    if word contains vowel then
        if word in lexicon then
            return lemma;
        else
            analyse POS tag;
            if word in irregulars then
                return lemma;
            else
                return lemma using Swedish morphology;
```

The lemmatizers reduced the number of word types by an average 29.87%: from 372,207 to 267,543 for Danish and from 403,867 to 276,188 for Swedish. Though unfortunately, the lemmatizers were not evaluated for their accuracy, because I had no gold standard to test them on. However, from what I saw, the results were acceptable, and having any lemmatizer would be better than having none. My expectation is, though, that the Danish lemmatizer has better results than the Swedish one, despite the Swedish one having a larger lexicon. This is partly because the Swedish lemmatizer relies more heavily on the morphological tags, which are not always that accurate.

4.1.2 Trimming

After lemmatization all words were stripped of their POS tags; this might have caused for a slight reduction of word types, with homonyms no longer being distinguished between by their POS tag. All texts were furthermore stripped of everything that is not a letter or a hyphen (as some words may contain a hyphen). All isolated hyphens were removed as well. After that, all lines that contained any symbols from other alphabets (Greek, Cyrillic, or even other), were removed, resulting in a file with 1,400,661 lines. This reduced the number of words and word types drastically, with an average of 35.79% for words and 29.96% for word types. The size of the reduction can be explained – partly – by the nature of the Europarl corpus: the corpus contains many names of laws and their codes. The codes contain many digits and numbers, which are all different word types and which were removed.

	Before	After	Reduction
English	52,835,267	35,162,510	33.45%
Danish	51,068,477	32,089,638	37.16%
German	52,411,025	32,694,553	37.62%
Dutch	55,182,552	35,813,093	35.10%
Swedish	49,557,249	31,901,254	35.63%

Table 4.2: Number of words before and after trimming per language.

	Before	After	Reduction
English	183,258	74,018[3]	59.61%
Danish	267,543	226,697	15.27%
German	340,087	248,656	26.88%
Dutch	281,403	193,979	31.07%
Swedish	276,188	229,375	16.95%

Table 4.3: Number of word types before and after trimming per language.

4.1.3 `cdec`'s word alignment algorithm

Dyer et al.'s (2010) `cdec` contains several tools for decoding, aligning and learning for statistical machine translation. It's word alignment tool, called `fast_align`, aligns two sentences on a word level. In word alignment words that correspond in meaning between two parallel sentences are aligned. In this research, word alignment was used to confine the number of word combinations (between the five languages) that the cognate recognition system should check cognacy for (see chapter 3). The `fast_align` tool was chosen for its speed (hence the name) and its efficient evaluation.

The `fast_align` tool was given the parallel language pairs with English as the source language and the other four languages as target languages. The output of the `fast_align` tool is a file with pairs of the indexes of the words that were aligned. These indexes then had to be translated into words. The result is a five-column file with lines containing the word alignments throughout the five languages. The file contains 35,162,510 lines, i.e. alignments, which is equal to the total number of English words. This is because the `fast_align` tool took English as the source language, finding alignments for every word.

4.1.4 Further preselection

Then, all lines that only occured once were removed; those are likely to be noise, and would greatly increase run time and decrease accuracy. In this process 33,599,374 alignments were removed, leaving 1,563,136.

Additionally, all lines in which the length of words differed too much were removed as well, because they are very unlikely to be cognates. The lines in which the length difference between the longest word and the shortest word in the line was greater than or equal to 5 letters were removed. As a final step all lines in which the Jaro-Winkler distance (Winkler, 1990) between the words in the line is smaller than 0.4 were removed, too. This was done so that lines with words that are too dissimilar will not be tested for cognacy, because it is very likely the words will not be cognates. This reduced the number of lines to 318,651 lines.

It was also planned to merge certain lines. Two lines for which in every column the two fields are identical or one of the two fields is empty, could be merged, as demonstrated in Figure 4.1. The newly formed line could then itself be subject to merging, such as in Figure 4.2. After merging, all subset lines were removed.

and		und		och
and	og	und	en	
and	og	und	en	och

Figure 4.1: The merging of two lines.

1.	and		und		
2.	and	og		en	
3.	and			en	och
1,2.	and	og	und	en	
3.	and			en	och
1,2,3.	and	og	und	en	och

Figure 4.2: The merging of three lines.

[3]It is striking to see how much fewer word types (after lemmatization and trimming) English uses, compared to the other languages. This has to do with the fact that English does not require compounds to be written as one word, whereas the other languages do.

1.	and		und		
2.	and	og		en	
3.	and			en	och
1.	and		und		
2.	and	og		en	
3.	and			en	och
1,2.	and	og	und	en	
1,3.	and		und	en	och
2,3.	and	og		en	och
1,2,3.	and	og	und	en	och

Figure 4.3: The merging of three lines with all possible combinations.

Due to an exponential blow-up this merging was not further pursued, as all newly formed lines through merging had to be considered for merging as well, such as in Figure 4.3. This resulted in millions of newly merged lines that had to be considered for merging with millions of other lines, leading to a memory overflow.[4] Although confining the number of lines for which the merger algorithm had to test for possible merging (e.g. lines that already existed were not created, and lines were stored in a clever way so that two lines containing two different German words, for example, were not tested) decreased run time significantly, it could not prevent a memory overflow.

4.1.5 A note on compounds

Compounds in the Europarl corpus raised a few issues. Because English was used as the source language for `fast_align`, some compounds in the other languages corresponded to only a part of the multi-word compound in English: for example, the Dutch word *belastingbeleid* 'taxation policy' would only correspond to either *taxation* or *policy*, while it should correspond to both words. Because of this issue it was considered using German as the source language, because German writes compounds as one word, and the one-word compound in Danish, Dutch and Swedish would then be aligned to the whole compound in German, instead of just to a part of the compound in English. Unfortunately though, the `fast_align` algorithm crashed every time another source language than English was used, probably due to a memory overflow (possibly because of the number of word types). Another solution would be to split all compounds in all languages (which would also happen to reduce the number of word types). The compound checker used for the Danish lemmatizer could have been used to do so. I, however, decided against compound splitting, as the results of compound splitting were too poor for all languages.

4.2 Training, development and test sets

For the training of the developed system an extended and slightly modified version of the Swadesh list (Swadesh, 1955) was used. The original Swadesh list – a list of words that are very common throughout languages, which is often used to compare languages – comprises 217 different words.

[4]On a machine with 8 GB of RAM. The memory overflow already occurred after processing a few hundred lines, out of the 318,651.

The modified version that was used for training purposes in this research is the Swadesh list for English joined with the Swadesh lists for Danish, German, Dutch, and Swedish.

The modified version of the Swadesh list was made in such a way that every line contains words sharing their etymon, i.e. cognates, throughout the five languages (e.g. the line '`I jeg ich ik jag`'). In the cases where a line of the joined Swadesh lists contained multiple etymons, the line was split into multiple lines, such that every line contained only one etymon, and if a language did not have a word from a certain etymon, the field was left blank (e.g. the line '`thou du du  du`', in which the fourth, i.e. the Dutch, field is empty). This is shown in Figure 4.4, in which the line containing only `jij`, was padded to '`ye i ihr jij i`'.

$$\texttt{thou}\quad\texttt{du}\quad\texttt{du}\quad\texttt{jij}\quad\texttt{du} \rightarrow \begin{cases} \texttt{thou}\quad\texttt{du}\quad\texttt{du}\qquad\qquad\texttt{du} \\ \qquad\qquad\qquad\quad\ \texttt{jij} \end{cases}$$

Figure 4.4: Splitting lines with two etymons.

If a language was the only one having a word from that etymon, the line was removed (among which, for example, the English *big* (Onions 1966: "of unkn[own] origin, possible Scand[inavian]"). In certain cases a cognate in one language can correspond to multiple cognates in the other; in these cases a new line was added for every cognate (e.g. the English *man* is cognates with both the German *Mann* 'man' and *man* 'one', resulting in the two lines '`man mand Mann man man`' and '`man mand man man man`' respectively).[5] The whole list was extended with a few lines of cognates that are not in the Swadesh list so that the list totalled 500 lines. Note that all words on one line do not necessarily mean the same thing. For the full extended, joined Swadesh list used for training purposes, see appendix C. Of this list, 60% was used for actual training.

The developed system does not need lines of non-cognate words for training purposes; in the training part, the system learns transition rules between languages, and calculates the probability of those rules given the cognates it saw, for example '`sk:sch:sk/`1.0' for Danish-German-Swedish, which means that in 100% of the cases a Danish <sk>, a German <sch> and a Swedish <sk> transition into each other, given each other.

The development set consisted of 20% of the extended Swadesh list, combined with the same amount of non-cognate lines. These lines were taken from a 1000 random lines from the result of the data preparation, which were removed from the full data. Lines that contained only cognates (and which should therefore be labelled `Y` by the system) were removed. This resulted in a number of lines that contained only non-cognate tuples and that should therefore be labelled `N`, meaning that not all words in that line are cognates of each other.

The test set had the same composition as the development set.

A training set, development set and test set were made for every combination of languages (ranging from size two to five, so 26 combinations in total), as the system was tested on all combinations. However, as not all lines had all fields filled, the length of the extended Swadesh list differed per combination. For example, if a line only contained Danish and Swedish, in any combination of languages that did not have either Danish or Swedish the resulting line contained only one language, in which case it was removed; in a combination that did not have both Danish and Swedish, the resulting line would be empty and thus removed as well. Any lines that became doubles because

[5] Do note that Dutch also has two cognates with the English *man*: *man* 'man' and *men* 'one'. This particular case would therefore result in four different lines ($1 \times 1 \times 2 \times 2 \times 1 = 4$), but for illustrative purposes that is ignored for now.

of the removal of some columns, were removed as well, so that all lines were unique. The length of the resulting extended Swadesh list influences the length of the training set, development set and test set. The lengths of the three sets for all combinations were as listed in table 4.4.

	Training	Dev.	Test
DA-DE	178	120	118
DA-NL	186	124	124
DA-SV	214	142	142
DE-NL	220	148	146
DE-SV	179	120	120
EN-DA	190	128	126
EN-DE	191	128	126
EN-NL	203	136	134
EN-SV	187	124	126
NL-SV	182	122	124
DA-DE-NL	237	158	158
DA-DE-SV	232	154	154
DA-NL-SV	238	160	158
DE-NL-SV	237	158	158
EN-DA-DE	241	160	162
EN-DA-NL	245	164	162
EN-DE-NL	241	160	160
EN-DA-SV	237	158	158
EN-DE-SV	235	156	158
EN-NL-SV	243	162	162
DA-DE-NL-SV	282	188	188
EN-DA-DE-NL	274	184	184
EN-DA-DE-SV	268	178	178
EN-DA-NL-SV	277	184	186
EN-DE-NL-SV	269	180	180
EN-DA-DE-NL-SV	300	200	200

Table 4.4: The length of the training set, development set and test set for all language combinations.

5. Probability theory

The transition probability of a tuple (or the confidence of the transition rule) can be compared to the transition probabilities of the subsets of the tuple. The transition probability of the tuple will be the observed probability; the expected probability is based on the probabilities of subsets of the tuple. In what follows I shall demonstrate the derivations of how the observed and expected probabilities can be compared.

5.1 Substring transitions

The observed probability of a rule $a \Rightarrow b$, in which a and b are substrings, graphemes, phonemes, phones, or sequences of those, is defined as the total number of occurrences where a transitions into b divided by the total number of occurrences of a, as illustrated in Equation 5.1.

$$P_o(a \Rightarrow b) = \frac{n(a \cap b)}{n(a)} \tag{5.1}$$

The observed probability of two substrings a and b transitioning into each other (i.e. a two-way transition) is then defined as the square root of the product of the probabilities of $a \Rightarrow b$ and $b \Rightarrow a$.[1] This is illustrated in Equation 5.2.

$$P_o(a : b) = \sqrt{\begin{aligned} &P_o(a \Rightarrow b) \\ &\quad \cdot P_o(b \Rightarrow a) \end{aligned}} = \sqrt{\frac{n(a \cap b)^2}{n(a) \cdot n(b)}} \tag{5.2}$$

The observed probabilities of three, four and five substrings transitioning into each other are similarly defined as in Equation 5.3, Equation 5.4 and Equation 5.5 respectively.

$$P_o(a : b : c) = \sqrt[3]{\begin{aligned} &P_o(a \Rightarrow b \cap a \Rightarrow c) \\ &\quad \cdot P_o(b \Rightarrow a \cap b \Rightarrow c) \\ &\quad \cdot P_o(c \Rightarrow a \cap c \Rightarrow b) \end{aligned}} = \sqrt[3]{\frac{n(a \cap b \cap c)^3}{n(a) \cdot n(b) \cdot n(c)}} \tag{5.3}$$

[1] I am not certain if this calculation actually leads to a true probability. The probabilities of substring transitions are rather pseudo-probabilities. In the remainder of the thesis, I shall continue calling them probabilities, but bear in mind that they are pseudo-probabilities, as it is unclear if they are true probabilities.

$$P_o(a:b:c:d) = \sqrt[4]{\begin{aligned} &P_o(a \Rightarrow b \cap a \Rightarrow c \cap a \Rightarrow d) \\ &\quad \cdot P_o(b \Rightarrow a \cap b \Rightarrow c \cap b \Rightarrow d) \\ &\quad \cdot P_o(c \Rightarrow a \cap c \Rightarrow b \cap c \Rightarrow d) \\ &\quad \cdot P_o(d \Rightarrow a \cap d \Rightarrow b \cap d \Rightarrow c) \end{aligned}} = \sqrt[4]{\frac{n(a \cap b \cap c \cap d)^4}{n(a) \cdot n(b) \cdot n(c) \cdot n(d)}} \tag{5.4}$$

$$P_o(a:b:c:d:e) = \sqrt[5]{\begin{aligned} &P_o(a \Rightarrow b \cap a \Rightarrow c \cap a \Rightarrow d \cap a \Rightarrow e) \\ &\quad \cdot P_o(b \Rightarrow a \cap b \Rightarrow c \cap b \Rightarrow d \cap b \Rightarrow e) \\ &\quad \cdot P_o(c \Rightarrow a \cap c \Rightarrow b \cap c \Rightarrow d \cap c \Rightarrow e) \\ &\quad \cdot P_o(d \Rightarrow a \cap d \Rightarrow b \cap d \Rightarrow c \cap d \Rightarrow e) \\ &\quad \cdot P_o(e \Rightarrow a \cap e \Rightarrow b \cap e \Rightarrow c \cap e \Rightarrow d) \end{aligned}} = \sqrt[5]{\frac{n(a \cap b \cap c \cap d \cap e)^5}{n(a) \cdot n(b) \cdot n(c) \cdot n(d) \cdot n(e)}} \tag{5.5}$$

When assuming independence of the transitions, the expected probabilities can be calculated from the probabilities of transitions of subsets of the transition tuple. For example, $P_o(a:b:c)$ can be calculated based on $P_o(a:b)$, $P_o(a:c)$ and $P_o(b:c)$. This is useful, as it allows us to compare expected and observed probabilities calculated for combinations of languages of different lengths: this we want to do to evaluate whether adding a language to the equation will actually help determining cognacy of words with a higher confidence. In what follows, I shall illustrate how substring-transition probabilities can be calculated with probabilities of subsets of the substring-transition tuple.

5.1.1 *P(a : b : c)*

$$\begin{aligned} P_e(a:b:c) &= \sqrt[3]{\begin{aligned} &P_o(a \Rightarrow b \cap a \Rightarrow c) \\ &\quad \cdot P_o(b \Rightarrow a \cap b \Rightarrow c) \\ &\quad \cdot P_o(c \Rightarrow a \cap c \Rightarrow b) \end{aligned}} \\ &= \sqrt[3]{\begin{aligned} &P_o(a \Rightarrow b) \cdot P_o(a \Rightarrow c) \\ &\quad \cdot P_o(b \Rightarrow a) \cdot P_o(b \Rightarrow c) \\ &\quad \cdot P_o(c \Rightarrow a) \cdot P_o(c \Rightarrow b) \end{aligned}} \end{aligned} \tag{5.6}$$

Given Equation 5.6:

- $P_e(a:b:c)$ can be calculated with two-way substring transitions as in Equation 5.7.

$$P_e(a:b:c) = \sqrt[3]{P_o(a:b)^2 \cdot P_o(a:c)^2 \cdot P_o(b:c)^2} \tag{5.7}$$

5.1.2 *P(a : b : c : d)*

$$\begin{aligned} P_e(a:b:c:d) &= \sqrt[4]{\begin{aligned} &P_o(a \Rightarrow b \cap a \Rightarrow c \cap a \Rightarrow d) \\ &\quad \cdot P_o(b \Rightarrow a \cap b \Rightarrow c \cap b \Rightarrow d) \\ &\quad \cdot P_o(c \Rightarrow a \cap c \Rightarrow b \cap c \Rightarrow d) \\ &\quad \cdot P_o(d \Rightarrow a \cap d \Rightarrow b \cap d \Rightarrow c) \end{aligned}} \\ &= \sqrt[4]{\begin{aligned} &P_o(a \Rightarrow b) \cdot P_o(a \Rightarrow c) \cdot P_o(a \Rightarrow d) \\ &\quad \cdot P_o(b \Rightarrow a) \cdot P_o(b \Rightarrow c) \cdot P_o(b \Rightarrow d) \\ &\quad \cdot P_o(c \Rightarrow a) \cdot P_o(c \Rightarrow b) \cdot P_o(c \Rightarrow d) \\ &\quad \cdot P_o(d \Rightarrow a) \cdot P_o(d \Rightarrow b) \cdot P_o(d \Rightarrow c) \end{aligned}} \end{aligned} \tag{5.8}$$

Given Equation 5.8:

- $P_e(a:b:c:d)$ can be calculated with two-way substring transitions as in Equation 5.9.

$$\begin{aligned} P_e(a:b:c:d) &= \sqrt[4]{\begin{aligned} &P_o(a:b)^2 \cdot P_o(a:c)^2 \cdot P_o(a:d)^2 \\ &\quad \cdot P_o(b:c)^2 \cdot P_o(b:d)^2 \cdot P_o(c:d)^2 \end{aligned}} \\ &= \sqrt{\begin{aligned} &P_o(a:b) \cdot P_o(a:c) \cdot P_o(a:d) \\ &\quad \cdot P_o(b:c) \cdot P_o(b:d) \cdot P_o(c:d) \end{aligned}} \end{aligned} \tag{5.9}$$

- $P_e(a:b:c:d)$ can be calculated with three-way substring transitions as in Equation 5.10.

$$
\begin{aligned}
P_e(a:b:c:d) &= \sqrt[8]{\begin{pmatrix} P_o(a \Rightarrow b) \cdot P_o(a \Rightarrow c) \cdot P_o(a \Rightarrow d) \\ \quad \cdot P_o(b \Rightarrow a) \cdot P_o(b \Rightarrow c) \cdot P_o(b \Rightarrow d) \\ \quad \cdot P_o(c \Rightarrow a) \cdot P_o(c \Rightarrow b) \cdot P_o(c \Rightarrow d) \\ \quad \cdot P_o(d \Rightarrow a) \cdot P_o(d \Rightarrow b) \cdot P_o(d \Rightarrow c) \end{pmatrix}^2} \\
&= \sqrt[8]{\begin{matrix} P_o(a \Rightarrow b \cap a \Rightarrow c) \cdot P_o(a \Rightarrow b \cap a \Rightarrow d) \\ \quad \cdot P_o(a \Rightarrow c \cap a \Rightarrow d) \cdot P_o(b \Rightarrow a \cap b \Rightarrow c) \\ \quad \cdot P_o(b \Rightarrow a \cap b \Rightarrow d) \cdot P_o(b \Rightarrow c \cap b \Rightarrow d) \\ \quad \cdot P_o(c \Rightarrow a \cap c \Rightarrow b) \cdot P_o(c \Rightarrow a \cap c \Rightarrow d) \\ \quad \cdot P_o(c \Rightarrow b \cap c \Rightarrow d) \cdot P_o(d \Rightarrow a \cap d \Rightarrow b) \\ \quad \cdot P_o(d \Rightarrow a \cap d \Rightarrow c) \cdot P_o(d \Rightarrow b \cap d \Rightarrow c) \end{matrix}} \\
&= \sqrt[8]{P_o(a:b:c)^3 \cdot P_o(a:b:d)^3 \cdot P_o(a:c:d)^3 \cdot P_o(b:c:d)^3}
\end{aligned} \tag{5.10}
$$

5.1.3 *P(a : b : c : d : e)*

$$
\begin{aligned}
P_e(a:b:c:d:e) &= \sqrt[5]{\begin{matrix} P_o(a \Rightarrow b \cap a \Rightarrow c \cap a \Rightarrow d \cap a \Rightarrow e) \\ \quad \cdot P_o(b \Rightarrow a \cap b \Rightarrow c \cap b \Rightarrow d \cap b \Rightarrow e) \\ \quad \cdot P_o(c \Rightarrow a \cap c \Rightarrow b \cap c \Rightarrow d \cap c \Rightarrow e) \\ \quad \cdot P_o(d \Rightarrow a \cap d \Rightarrow b \cap d \Rightarrow c \cap d \Rightarrow e) \\ \quad \cdot P_o(e \Rightarrow a \cap e \Rightarrow b \cap e \Rightarrow c \cap e \Rightarrow d) \end{matrix}} \\
&= \sqrt[5]{\begin{matrix} P_o(a \Rightarrow b) \cdot P_o(a \Rightarrow c) \cdot P_o(a \Rightarrow d) \cdot P_o(a \Rightarrow e) \\ \quad \cdot P_o(b \Rightarrow a) \cdot P_o(b \Rightarrow c) \cdot P_o(b \Rightarrow d) \cdot P_o(b \Rightarrow e) \\ \quad \cdot P_o(c \Rightarrow a) \cdot P_o(c \Rightarrow b) \cdot P_o(c \Rightarrow d) \cdot P_o(c \Rightarrow e) \\ \quad \cdot P_o(d \Rightarrow a) \cdot P_o(d \Rightarrow b) \cdot P_o(d \Rightarrow c) \cdot P_o(d \Rightarrow e) \\ \quad \cdot P_o(e \Rightarrow a) \cdot P_o(e \Rightarrow b) \cdot P_o(e \Rightarrow c) \cdot P_o(e \Rightarrow d) \end{matrix}}
\end{aligned} \tag{5.11}
$$

Given Equation 5.11:

- $P_e(a:b:c:d:e)$ can be calculated with two-way substring transitions as in Equation 5.12.

$$
P_e(a:b:c:d:e) = \sqrt[5]{\begin{matrix} P_o(a:b)^2 \cdot P_o(a:c)^2 \cdot P_o(a:d)^2 \cdot P_o(a:e)^2 \cdot P_o(b:c)^2 \\ \quad \cdot P_o(b:d)^2 \cdot P_o(b:e)^2 \cdot P_o(c:d)^2 \cdot P_o(c:e)^2 \cdot P_o(d:e)^2 \end{matrix}} \tag{5.12}
$$

- $P_e(a:b:c:d:e)$ can be calculated with three-way substring transitions as in Equation 5.13.

$$
\begin{aligned}
P_e(a:b:c:d:e) &= \sqrt[15]{\begin{pmatrix} P_o(a \Rightarrow b) \cdot P_o(a \Rightarrow c) \cdot P_o(a \Rightarrow d) \cdot P_o(a \Rightarrow e) \\ \cdot P_o(b \Rightarrow a) \cdot P_o(b \Rightarrow c) \cdot P_o(b \Rightarrow d) \cdot P_o(b \Rightarrow e) \\ \cdot P_o(c \Rightarrow a) \cdot P_o(c \Rightarrow b) \cdot P_o(c \Rightarrow d) \cdot P_o(c \Rightarrow e) \\ \cdot P_o(d \Rightarrow a) \cdot P_o(d \Rightarrow b) \cdot P_o(d \Rightarrow c) \cdot P_o(d \Rightarrow e) \\ \cdot P_o(e \Rightarrow a) \cdot P_o(e \Rightarrow b) \cdot P_o(e \Rightarrow c) \cdot P_o(e \Rightarrow d) \end{pmatrix}^3} \\
&= \sqrt[15]{\begin{matrix} P_o(a:b:c)^3 \cdot P_o(a:b:d)^3 \cdot P_o(a:b:e)^3 \cdot P_o(a:c:d)^3 \\ \cdot P_o(a:c:e)^3 \cdot P_o(a:d:e)^3 \cdot P_o(b:c:d)^3 \\ \cdot P_o(b:c:e)^3 \cdot P_o(b:d:e)^3 \cdot P_o(c:d:e)^3 \end{matrix}} \\
&= \sqrt[5]{\begin{matrix} P_o(a:b:c) \cdot P_o(a:b:d) \cdot P_o(a:b:e) \cdot P_o(a:c:d) \\ \cdot P_o(a:c:e) \cdot P_o(a:d:e) \cdot P_o(b:c:d) \\ \cdot P_o(b:c:e) \cdot P_o(b:d:e) \cdot P_o(c:d:e) \end{matrix}}
\end{aligned}
\tag{5.13}
$$

- $P_e(a:b:c:d:e)$ can be calculated with four-way substring transitions as in Equation 5.14.

$$
\begin{aligned}
P_e(a:b:c:d:e) &= \sqrt[15]{\begin{pmatrix} P_o(a \Rightarrow b) \cdot P_o(a \Rightarrow c) \cdot P_o(a \Rightarrow d) \cdot P_o(a \Rightarrow e) \\ \cdot P_o(b \Rightarrow a) \cdot P_o(b \Rightarrow c) \cdot P_o(b \Rightarrow d) \cdot P_o(b \Rightarrow e) \\ \cdot P_o(c \Rightarrow a) \cdot P_o(c \Rightarrow b) \cdot P_o(c \Rightarrow d) \cdot P_o(c \Rightarrow e) \\ \cdot P_o(d \Rightarrow a) \cdot P_o(d \Rightarrow b) \cdot P_o(d \Rightarrow c) \cdot P_o(d \Rightarrow e) \\ \cdot P_o(e \Rightarrow a) \cdot P_o(e \Rightarrow b) \cdot P_o(e \Rightarrow c) \cdot P_o(e \Rightarrow d) \end{pmatrix}^3} \\
&= \sqrt[15]{\begin{matrix} P_o(a:b:c:d)^4 \cdot P_o(a:b:c:e)^4 \cdot P_o(a:b:d:e)^4 \\ \cdot P_o(a:c:d:e)^4 \cdot P_o(b:c:d:e)^4 \end{matrix}}
\end{aligned}
\tag{5.14}
$$

5.1.4 Generalization

These expected probabilities can be generalized as put forth in 5.15.

$$\begin{cases} \text{Let } \Sigma \text{ be the set of all substrings in the transition tuple;} \\ \text{Let } C_t \text{ be the set of all combinations of length } t \text{ of } \lambda \in \Sigma; \\ s = |\Sigma|; \qquad c = |C_t|; \qquad s > t > 1; \\ \text{Then:} \\ \hline P_e(\lambda_1 : ... : \lambda_s) = \left(\prod_{i=1}^{c} P_o(\gamma_i \in C_t) \right)^E \quad \text{with } E = \dfrac{(s-1)}{(t-1) \cdot \binom{s}{t}} \end{cases} \tag{5.15}$$

For example, $P_e(a : b : c : d)$ based on three-way substring transitions would be:

$$\begin{cases} \Sigma = \{a, b, c, d\}; \\ C_t = \{a : b : c, a : b : d, a : c : d, b : c : d\}; \\ s = |\Sigma| = 4; \qquad c = |C_t| = 4; \qquad t = 3; \\ \text{Then:} \\ \hline P_e(\lambda_1 : ... : \lambda_s) = \left(\prod_{i=1}^{4} P_o(\gamma_i \in C_t) \right)^E \quad \text{with } E = \dfrac{(4-1)}{(3-1) \cdot \binom{4}{3}} = \dfrac{3}{8} \\ P_e(a : b : c : d) = \Big(P_o(a : b : c) \cdot P_o(a : b : d) \cdot P_o(a : c : d) \cdot P_o(b : c : d) \Big)^{\frac{3}{8}} \end{cases} \tag{5.16}$$

Notice that the result of Equation 5.16 equals Equation 5.10, despite a slightly different notation.

5.2 String transitions

The probability of words Λ_1 to Λ_s being cognates is defined as the n^{th} root of the iterated product of the probabilities of all aligned substring transitions λ_1 to λ_n, where n is the number of substring partitions (i.e. the length of the substring alignment (e.g. Figure 3.1)). See Equation 5.17. The n^{th} root is taken of the product to correct for the length of the alignments, so that longer words do not necessarily have a (much) lower probability of being cognates.

$$P(\Lambda_1 : ... : \Lambda_s) = \sqrt[n]{\prod_{i=1}^{n} P(\lambda_{1i} : ... : \lambda_{si})} \tag{5.17}$$

For example:

$$P(A : B) = \sqrt[n]{\prod_{i=1}^{n} P(a_i : b_i)} \tag{5.18}$$

Though it was shown in section 5.1 that the expected probabilities of all s-way substring transitions can be calculated based on the observed probabilities of t-way substring transitions ($s > t$), it cannot unthinkingly be concluded that Equation 5.7, Equation 5.9, Equation 5.10, Equation 5.12, Equation 5.13 and Equation 5.14 hold for words Λ_1 to Λ_s as well. In what follows I shall proof that Equation 5.7 holds for words as well. For the proofs that the other equations hold for words as well, I refer to appendix D.

$$P(A:B:C) = \sqrt[n]{\prod_{i=1}^{n} P(a_i:b_i:c_i)} \tag{5.19}$$

Given Equation 5.18 and Equation 5.19, $P_e(A:B:C)$ can be calculated with two-way word transitions, given Equation 5.7. Equation 5.20 proves that Equation 5.7 applies to words as well, and not only to substrings.

$$\begin{aligned}
P_e(A:B:C) &= \sqrt[n]{\prod_{i=1}^{n} \sqrt[3]{P_o(a_i:b_i)^2 \cdot P_o(a_i:c_i)^2 \cdot P_o(b_i:c_i)^2}} \\
&= \sqrt[n]{\prod_{i=1}^{n} \Big(P_o(a_i:b_i) \cdot P_o(a_i:c_i) \cdot P_o(b_i:c_i)\Big)^{\frac{2}{3}}} \\
&= \sqrt[n]{\left(\prod_{i=1}^{n} P_o(a_i:b_i) \cdot P_o(a_i:c_i) \cdot P_o(b_i:c_i)\right)^{\frac{2}{3}}} \\
&= \left(\prod_{i=1}^{n} P_o(a_i:b_i) \cdot P_o(a_i:c_i) \cdot P_o(b_i:c_i)\right)^{\frac{2}{3n}} \\
&= \left(\prod_{i=1}^{n} P_o(a_i:b_i) \cdot \prod_{i=1}^{n} P_o(a_i:c_i) \cdot \prod_{i=1}^{n} P_o(b_i:c_i)\right)^{\frac{2}{3n}} \\
&= \sqrt[3]{\left(\prod_{i=1}^{n} P_o(a_i:b_i) \cdot \prod_{i=1}^{n} P_o(a_i:c_i) \cdot \prod_{i=1}^{n} P_o(b_i:c_i)\right)^{\frac{2}{n}}} \\
&= \sqrt[3]{\left(\prod_{i=1}^{n} P_o(a_i:b_i)\right)^{\frac{2}{n}} \cdot \left(\prod_{i=1}^{n} P_o(a_i:c_i)\right)^{\frac{2}{n}} \cdot \left(\prod_{i=1}^{n} P_o(b_i:c_i)\right)^{\frac{2}{n}}}
\end{aligned} \tag{5.20}$$

$$
\begin{aligned}
P_e(A:B:C) &= \sqrt[3]{\left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i)}\right)^2 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:c_i)}\right)^2 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(b_i:c_i)}\right)^2} \\
&= \sqrt[3]{P_o(A:B)^2 \cdot P_o(A:C)^2 \cdot P_o(B:C)^2}
\end{aligned} \tag{5.20}
$$

5.2.1 Generalization

Given Equation 5.15 the calculation of the expected probability of word transitions can be generalized as in 5.21:

$$
\begin{cases}
\text{Let } \Sigma_{\lambda i} \text{ be the set of substrings in substring-transition tuple } i; \\
\text{Let } \Sigma_{\Lambda} \text{ be the set of words in the transition tuple;} \\
\text{Let } C_{\lambda t i} \text{ be the set of all combinations of length } t \text{ of } \lambda \in \Sigma_{\lambda i}; \\
\text{Let } C_{\Lambda t} \text{ be the set of all combinations of length } t \text{ of } \Lambda \in \Sigma_{\Lambda}; \\
s = |\Sigma_{\lambda i}| = |\Sigma_{\Lambda}|; \qquad c = |C_{\lambda t i}| = |C_{\Lambda t}|; \qquad s > t > 1; \\
\text{Then:} \\
\hline
P_e(\Lambda_1 : ... : \Lambda_s) = \sqrt[n]{\prod_{i=1}^{n} \left(\prod_{j=1}^{c} P_o(\gamma_j \in C_{\lambda t i}) \right)^E} \quad \text{with } E = \dfrac{(s-1)}{(t-1) \cdot \binom{s}{t}} \\
\text{This equals:} \\
P_e(\Lambda_1 : ... : \Lambda_s) = \left(\prod_{i=1}^{c} P_o(\Gamma_i \in C_{\Lambda t}) \right)^E \quad \text{with } E = \dfrac{(s-1)}{(t-1) \cdot \binom{s}{t}}
\end{cases} \tag{5.21}
$$

If, now, the observed and expected probabilities are not equal, that means that the transitions are not independent. If the observed probability is higher than the expected probability, that means that having more languages leads to more confident conclusions of words being cognates.

6. Description of the systems

In short, the machine learning system developed for cognate recognition first counts how often substrings occur together between languages in a given list of cognate tuples (the training set; see section 4.2). These combinations of substrings, which are considered possible substring-transition rules, are given weights. It then tries to find, using these weights, the best substring alignment for every cognate tuple, and calculates the probabilities of the transitions it has found in the alignments (using the formulae in Equation 5.2, Equation 5.3, Equation 5.4 and Equation 5.5). When having established the probabilities of the rules, the system uses the development set to determine a probability threshold. All tuples for which a probability lower than the threshold is calculated, will receive the label `N`, meaning that the words in the tuple are not cognates – all tuples for which a probability higher than the threshold is calculated, will receive the label `Y`, meaning that the words in the tuple are cognates. The system then uses the test set to evaluate the performance of the system.

The system that extracts the cognates from the list of possible cognates (see section 4.1) will use the determined threshold to extract the tuples in the list that are cognates, and saves them to a file.

In what follows I shall describe the two systems in more detail: first the machine learning system, then the extraction system.

6.1 Machine learning system

6.1.1 Weighting

The first line of the training-set file, the development-set file and the test-set file contains the languages each column represents. This is important so that the system knows which languages it is processing. The first thing, therefore, the system does is saving the languages.

The system then reads the training file, in which all words in the file are padded with the word-boundary symbols '`^`' and '`$`'. Furthermore all double consonants are replaced by single consonants. This is done to account for spelling differences that may occur between languages. For example, Dutch and German often use double consonants, even though they are pronounced the same way as single consonants.[1] Admittedly in a few rare cases, this would generate errors, such as English *but* and *butt*, but this was not considered a problem. All lines in the file are saved as tuples.

For every tuple the system compresses the tuple if there are empty fields, so that all fields in the tuple are filled. In doing so, it remembers which languages all words in the tuple belong to. For every word in the tuple, it then makes a set of all possible partitions of the word, taking into account the possibility of an empty string existing between partitions. Consider Figure 6.1, in which the word on the first line results in the possible partitions on the second line, which in turn result in the possible ϵ-partitions (partitions with empty strings) as on the third line – all partitions

[1]This double-to-single-consonant reduction is not done iteratively: triple consonants are therefore reduced to double consonants. This is not a problem, though, as triple consonants are not attested in the data. In fact, it could only occur in German compounds where the first part ends in a double consonant and the second part starts with the same consonant, which is extremely rare.

on the second and the third line are saved in a set. This results in a exponential blow-up, the longer the strings are, with $N_\epsilon = 3^{n+1}$, where n is the length of the string. In order to reduce the number of possible ϵ-partitions, partitions with parts containing both vowels and consonants were taken out, as well as those in which two following parts in the partition (either with or without an ϵ between them) contained vowels or semivowels. Graphemes which may represent semivowels (like <y> or <w>) were allowed in the same part only if they followed a grapheme representing a vowel (e.g. <oy>); if they preceded the vowel, I considered they were representing (semi-)consonants (e.g. <yo>). This was done so that diphthongs are always treated as a unit.

`^å$`

[^å$]	[^,å$]	[^å,$]	[^,å,$]
([^å$])	([^,å$] [^,ϵ,å$])	([^å,$] [^å,ϵ,$])	([^,å,$] [^,ϵ,å,$] [^,å,ϵ,$] [^,ϵ,å,ϵ,$])

Figure 6.1: ϵ-partitions of the Swedish word *å* 'river'. The four partitions on the second line result in the ϵ-partitions on the third line.

The ϵ-partitions are then grouped based on their length for each language. Then for every length, the system counts for every combination of partitions of that length between languages (so, a combination of a partition of length l of the word of language L_1, and a partition of length l of the word of language L_2, up to L_n) how often the aligned substrings occur together (the possible substring transitions), and then adds that up to what it has found earlier, for every combination of languages. So for example, in Figure 6.2 all lines are a possible partition for a word in the tuple. It then adds 1 to the count of <o>, <u> and <oe> occurring together in English, German and Dutch; it adds 1 to the count of <o> and <u> occurring together in English and German; it adds 1 to the count of <o> and <oe> occurring together in English and Dutch; it adds 1 to the count of <u> and <oe> occurring together in German and Dutch; etc. for every possible substring transition. It does this for every possible combination of ϵ-partitions, for every length, for every tuple.

In order to reduce the number of possible ϵ-partition combinations (which grows exponentially with the length of the strings with $N_{C\epsilon} = 3^{L(n+1)}$, where n is the length of the strings and L is the number of languages), the system does not consider those in which any of the possible substring transitions contains only empty strings. That is because substring alignments in which one of the substring transitions contains only empty strings (ϵ goes to ϵ) are considered wrong or redundant: the substring alignment could then as well be done without the ϵ-to-ϵ transition.

The result is a dictionary with counts of every possible substring transition for every language combination. The substrings that occur more often together (on more or less the same position) between the languages will have higher counts, because if, for example, every word in a tuple contains a <b>, and every word in another tuple contains a <b> as well, but in only one tuple all <b>s are followed by an <r>, the possible substring transition `br:br:br` will have lower counts than the possible substring transition `b:b:b`. Especially those transitions that are very unlikely will have very low counts, such as the possible substring transition `br:ϵ:oe`.

After having counted all the possible substring transitions, the system divides the number of times a possible substring transition occurs (i.e. the number of times the substrings occur together;

b	r	o	th	e	r
B	r	u	d	e	r
b	r	oe	d	e	r

$\Downarrow$

EN-DE-NL		EN-DE		EN-NL		DE-NL	
Add:	Nr.	Add:	Nr.	Add:	Nr.	Add:	Nr.
`b:b:b`	1	`b:b`	1	`b:b`	1	`b:b`	1
`r:r:r`	2	`r:r`	2	`r:r`	2	`r:r`	2
`o:u:oe`	1	`o:u`	1	`o:oe`	1	`u:oe`	1
`th:d:d`	1	`th:d`	1	`th:d`	1	`d:d`	1
`e:e:e`	1	`e:e`	1	`e:e`	1	`e:e`	1

Figure 6.2: An example of substring alignment for an English, German and Dutch word. All possible substring transitions are counted, and added to the counts found earlier for the particular transitions.

e.g. $n(\mathtt{b}_{\mathrm{EN}} \cap \mathtt{b}_{\mathrm{DE}} \cap \mathtt{b}_{\mathrm{NL}}))$ by the sum of the occurrences of the substrings in each individual language (e.g. $n(\mathtt{b}_{\mathrm{EN}}) + n(\mathtt{b}_{\mathrm{DE}}) + n(\mathtt{b}_{\mathrm{NL}})$). This value is multiplied by the number of languages, so that it is adjusted for the relatively larger denominator the more languages there are. The resulting value is the actual weight (not the possibility!) of the possible substring-transition rule. The calculation of the weight can be generalized as in Equation 6.1, in which r is the possible substring-transition rule, a is a substring that is part of the possible substring transition, l is the number of languages and $n(x)$ the number of occurrences of x.

$$w_r = l \cdot \frac{n\left(\bigcap_{i=1}^{l} a_i\right)}{\sum_{j=1}^{l} n(a_j)} \tag{6.1}$$

The weights and possible substring-transition rules are then written to a file, except for those in which all substrings in the possible substring-transition rule are a word-boundary symbol. These word-boundary symbols in a sense also denote an empty string, and similarly to the substring alignments with an ϵ-to-ϵ transition, substring alignments with a word-boundary-to-word-boundary transition are considered wrong. Therefore the possible substring-transition rules with only word-boundary symbols are removed, as they assign too much weight to wrong substring alignments. Furthermore, all possible substring-transition rules in which one substring contains both word-boundary symbols are removed as well, because these can only occur with one-letter words, which are very few in number (two in English, four in Danish, one in German, two in Dutch and four in Swedish ("jalu.ch – One letter words," 2016)), and will make the weighting more noisy.

In pseudo-code, the weighting algorithm can be summarized to Algorithm 4.

Algorithm 4: Possible substring-transition rule weighter

```
Result: Calculate a weight for every possible substring-transition rule
read languages as L;
read training set file as T;
for tuple in T do
    for word in tuple do
        pad word with word-boundary symbols;
        reduce double consonants in word to one;
    compress so that tuple has no empty fields;
    compress same fields in L as L';
    for word in tuple do
        make set of ϵ-partitions of word as S;
        group ϵ-partitions in S on length as S_l;
    for every length l do
        for combination in S_l1 × ... × S_ln do
            for possible substring transition in combination as PST do
                for combination of languages in C(L') as c do
                    n(PST) add 1
                for substring in PST do
                    n(substring) add 1 for corresponding language;
for every PST do
    weight = |L'| · n(PST)÷sum n(substring) for substring in PST
```

6.1.2 Substring alignment

The substring aligner, just as the weighter, reads the languages of the training file and then reads the training file itself in the same way as the weighter (padding words with word-boundary symbols and reducing double consonants). All lines are saved as tuples. Different to the weighter, though, is that the substring aligner reads the possible substring transitions and their weights, which were calculated by the weighter. It reads all possible substring transitions and weights for every language combination, and saves everything to a dictionary.

Then the substring aligner tries to find the best substring alignment for every tuple in the training set. In order to do this, it first compresses the tuple so that it has no empty fields, and remembers which languages remain. It then changes every word in the tuple to the longest possible ϵ-partition, again with diphthongs treated as a unit. After doing so, the system will consider every possible vowel alignment – for example, for the cognate tuple `church:Kirche:kerk`, the possible vowel alignments are `u:i:e` and `u:e:e`, as illustrated in Figure 6.3. Every possible vowel alignment had to be considered as it sometimes led to wrong alignments if it did not do so.

`ch`	`u`	`rch`		`ch`	`u`	`rch`
`K`	`i`	`rche`	and	`Kirch`	`e`	ϵ
`k`	`e`	`rk`		`k`	`e`	`rk`

Figure 6.3: The possible vowel alignments for the cognate tuple `church:Kirche:kerk`.

These vowel alignments produce a three-way split of the strings, as can be seen in 6.3: the first part is `ch:K:k`, the second part is `u:i:e` and the third part is `rch:rche:rk` (for the first vowel alignment). For every part it makes ϵ-partitions, and for every combination of ϵ-partitions of the same length (having the same restrictions as the weighter in that it does not consider combinations in which there are ϵ-to-ϵ transitions) it calculates a weight for the alignment by multiplying all weights of the substring transitions (with a default of 0 if the substring transition is not attested), and then taking the n^{th} root of the product, where n is the length of the substring alignment. The substring alignment of the part that has the highest weight will be considered the most likely. Every part, then, has a weighted substring alignment. However, to improve run speed (reducing the run time from days, possibly weeks, due to an exponential blow-up, to several hours), the first and the third part are cut off after five characters. For example, if the Dutch word `jurisprudentie` is aligned on the `u`, the parts would be cut off to `rispr` and `denti`. See Figure 6.4 for an illustration.

```
jurispr  u  dentie
  rispr  u  denti
```

Figure 6.4: An illustration of the cut-off as performed on the first and third part after vowel alignment.

The weight of the substring alignment of all parts combined (i.e. the whole words) are calculated by multiplying the weights of the substring alignments of the individual parts. This is done for every possible vowel alignment; the substring alignment with the highest weight will then be the substring alignment for that cognate tuple.

Now that all cognate tuples have been aligned on a substring level, probabilities (and not weights) are calculated for every substring transition. Every substring transition in every aligned cognate tuple will be counted for every language combination, as well as the occurrences of substrings within one language. The probabilities for all substring-transition rules will then be calculated with the formulae as presented in Equation 5.2, Equation 5.3, Equation 5.4 and Equation 5.5 in section 5.1. The rules, along with their probabilities, will then be written to a file.

In pseudocode, the substring aligning algorithm and probability calculation algorithm can be summarized to Algorithm 5.

Algorithm 5: Substring aligner and substring-transition-rule probability calculator

Result: Calculate a probability for substring-transition rules
read languages as L;
read training set file as T;
read weights as W;
for *tuple in* T **do**
  | find best substring alignment for tuple, given W;
for *aligned tuple in* T **do**
  | **for** *substring transition in tuple* **do**
  | | **for** *combination of languages* **do**
  | | | n(substring transition) add 1;
  | | **for** *substring in substring transition* **do**
  | | | n(substring) add 1 for corresponding language;
calculate probability for substring transitions using Equation 5.2, Equation 5.3, Equation 5.4 and Equation 5.5;
return substring-transition with probability

6.1.3 Threshold determination

The probability threshold beyond which words in tuples are considered cognates is calculated using the development set. The development set is read in the same fashion as the weighter and the substring aligner read the training set. However, one major difference is that the development set contains tuples the words of which are not cognates as well. This is dealt with by splitting the data set in cognate tuple and non-cognate tuples and saving them to different lists. Rules are read in the same way as the weights were read for the substring aligner.

For both cognate tuples and non-cognate tuples, the system first compresses the tuple so that it has no empty fields, and remembers which languages remain. Then it tries to find the best substring alignment for the words in the tuple in the same way as the substring aligner, only this time the probabilities of the substring-transition rules are used instead of the weights. Another difference is that the threshold determiner uses some sort of smoothing, so that substring alignments with unattested substring transitions do not necessarily have a probability of 0. If a substring transition is unattested, the system will not return a probability of 0, but rather a probability of $\frac{1}{1+V}$ in which V is the number of substring types found in the training data for every language in the tuple. This is loosely based on Laplace smoothing and gives rather good results. However, if the substring transition is not attested, but all substrings involved in the transition are letters representing vowels or diphthongs or all substrings involved are the same letter, the system returns $\frac{2}{1+V}$, so that vowels are more likely to transition into vowels and substrings are more likely to transition into themselves. The probability of a substring alignment is then calculated by multiplying all probabilities of the substring-transitions, and taking the n^{th} root of the product, in which n is the length of the substring alignment. The alignment with the highest probability will be returned as the best alignment: the probability of the alignment is the probability of the words in the tuple being cognates.

Now that it knows the probability for all cognate tuples and non-cognate tuples, the system will determine a probability threshold (all tuples with a higher probability than the threshold are cognates, all tuples with a lower probability than the threshold are not) by finding the probability for which as many cognate tuples as possible have a higher probability and at the same time as many non-cognate tuples as possible have a lower probability. Therefore the probability for which the distance between the number of false positives (tuples of non-cognates that have a higher probability than the threshold) and the number of false negatives (tuples of cognates that have a lower probability than the threshold) is the smallest, is the probability threshold that should bear the best results. The distance between the number of false positives and the number of false negatives is defined as in Equation 6.2.

$$\Delta = |FN^2 - FP^2| \tag{6.2}$$

For the final evaluation of the system, it reads the test set and calculates a probability for every tuple in the same manner as was done with the development set. Given the newly calculated threshold, it calculates the number of true positives (tuples of cognates that have a higher probability than the threshold), false positives (tuples of non-cognates that have a higher probability than the threshold), false negatives (tuples of cognates that have a lower probability than the threshold) and true negatives (tuples of non-cognates that have a lower probability than the threshold), and returns an accuracy, precision, recall and f-score based on those numbers.

In pseudocode, the probability-threshold calculator and evaluation system can be summarized to Algorithm 6.

Algorithm 6: Probability-threshold calculator and evaluator

```
Result: Calculate threshold, evaluate system
read development set file as D;
read rules as R;
for Y and N labels in D do
    for tuple in label list do
        find best substring alignment for tuple, given R and smoothing;
        remember probability;
for threshold between 0 and 1, step= 0,001 do
    calculate number of false positives;
    calculate number of false negatives;
    calculate distance;
threshold = threshold with minimal distance;
read test set file as T;
for Y and N labels do
    for tuple in label list in T do
        find best substring alignment for tuple, given R and smoothing;
        if label is Y then
            if P(alignment)≥threshold then
                TP add 1;
            else
                FN add 1;
        else
            if P(alignment)≥threshold then
                FP add 1;
            else
                TN add 1;
return accuracy, precision, recall, f-score
```

6.2 Extraction system

The cognate-extraction system which was run on the list of possible cognates uses the same probability calculation as the probability-threshold calculator and evaluation system use, and uses the threshold that was determined earlier. For every tuple of possible cognates, it calculates the probability that all words in the tuple are cognates of each other. If the calculated probability is higher than the threshold (which differs for every language combination the system is run on), the extraction system writes the tuple to a file. In pseudo-code this can be summarized to Algorithm 7 on page 30.

Algorithm 7: Cognate-extraction system

Result: Extract cognate tuple from file
read data as D;
read rules as R;
read threshold as t;
for *tuple in* D **do**
    calculate probability;
    if *probability* $\geq t$ **then**
        write tuple to file

7. Results

In order to compare results, all combinations of the five languages (ranging from two to five languages) were processed by the transition-rules learner, the cognate-recognition system and an SVM. In what follows all results will be discussed for those three systems, as well as the observed probabilities as opposed to the expected probabilities.

7.1 Transition-rules learner

The transition-rules learner found 37,098 rules in total throughout the 26 language combinations. It was found that, as the number of languages in the combinations grew, the number of rules that had a probability of 1.0 (i.e. 100%) declined. In the same manner the means of the probabilities of all rules per combination declined. The average of the means of all probabilities for the transition rules for two languages was 0.485, whereas for three, four and five languages it was 0.287, 0.233 and 0.179 respectively. The average means and their decline are shown in Figure 7.1.

Nr. of languages	Average mean
2	0.485
3	0.287
4	0.223
5	0.179

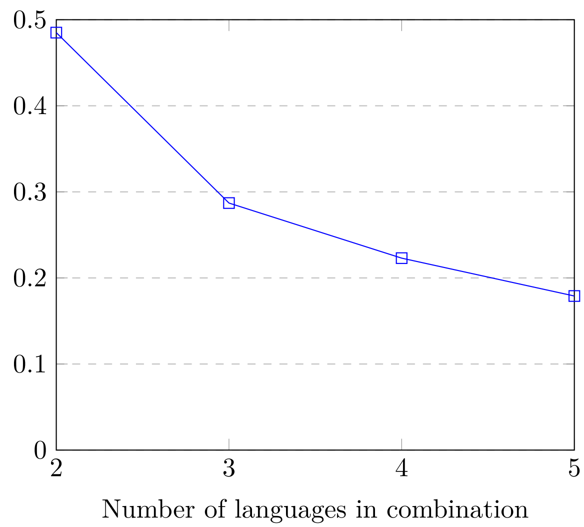

Figure 7.1: The average means of the found rules for the language combinations.

Figure 7.2 and Figure 7.3, for illustration, show the first ten rules (which happen to have a 100% probability) for the combination Danish-Swedish and for the combination with all five languages respectively. Despite the system ignoring partitions with parts containing both letters representing a consonant and letters representing a vowel, there are rules that contain letters representing a consonant and a `y`. This is because the letter <y> is defined for the system as both a semivowel and a vowel, and a semivowel is treated the same as a consonant. Nonetheless, these rules are rather precise. Remember that the symbols `ˆ` and `$` denote word boundaries.

Rule	P
`vn$:mn$`	1.0
`^t:^t`	1.0
`^å$:^å$`	1.0
`gt$:kt$`	1.0
`^k:^k`	1.0
`^ryg:^ryk`	1.0
`^h:^h`	1.0
`rp$:rp$`	1.0
`m:m`	1.0
`^kl:^kl`	1.0

Figure 7.2: The first ten rules of the combination Danish-Swedish.

Rule	P
`rn$:rn$: rn$ :rn$:rn$`	1.0
`ls : ls :  ls  : ls : ls`	1.0
`^kn:^kn: ^kn :^kn:^kn`	1.0
`ly$:lg$: lg$ :lg$:lg$`	1.0
`rm$:rm$: rm$ :rm$:rm$`	1.0
`^gl:^gl: ^gl :^gl:^gl`	1.0
`^sn:^sn:^schn:^sn:^sn`	1.0
`^br:^br: ^br :^br:^br`	1.0
`mp$:mp$: mpf$ :mp$:mp$`	1.0
`^bl:^bl: ^bl :^bl:^bl`	1.0

Figure 7.3: The first ten rules of the combination with all five languages.

7.2 Observed and expected probabilities

For every combination of languages, the system also calculated the mean of the probabilities assigned to all cognate tuples in the extended Swadesh list for that combination – which is the average of the observed probabilities. For every combination, the system also calculated the mean of the expected probabilities using the formulae in chapter 5.

It was found that the observed probabilities were higher than the expected probabilities. This means that the decision whether a tuple contains cognates or not is more precise (or more confident) when using transition rules for more languages, than when combining the calculated probabilities of language combinations with fewer languages. For example, the probability that English *book* and German *Buch* are cognates is found to be 0.498 ($P_o(\texttt{book}_{\text{EN}} : \texttt{Buch}_{\text{DE}}) = 0.498$). For Dutch *boek* probabilities are found to be $P_o(\texttt{book}_{\text{EN}} : \texttt{boek}_{\text{NL}}) = 0.540$ and $P_o(\texttt{Buch}_{\text{DE}} : \texttt{boek}_{\text{NL}}) = 0.679$. The expected probability (calculated using Equation 5.7) that all three words are cognates of each other is $\sqrt[3]{0.498^2 \cdot 0.540^2 \cdot 0.679^2} = 0.322$. However, the observed probability that all three words are cognates of each other is 0.485. Therefore using transition rules for more languages results in more confident decisions. The means of the observed and expected probabilities for the language combinations of length n are shown in Table 7.1 on page 33.

Run-time grew exponentially with about a minute for two languages, about six and a half minutes for three languages, about an hour for four languages and about eighteen hours for five languages.

7.3 Cognate-recognition system

The cognate recognition system was evaluated on the test set, as explained in section 6.1.3. The average accuracy, precision, recall and f-score were highest with language pairs, and decreased as the number of languages increased. The average f-score for language pairs was 0.850; for language triples it was 0.819; and for language quadruples it was 0.773. The calculated probability thresholds went down as the number of languages grew – this was to be expected as the average of the probabilities of the transition rules went down as the number of languages increased as well. See Table 7.2 for an overview of all average numbers; see Table 7.3 on page 34 for all numbers.

		P_e		
n	P_o	$n-1$	$n-2$	$n-3$
2	0.478			
3	0.256	0.211		
4	0.174	0.134	0.099	
5	0.120	0.091	0.065	0.0003

Table 7.1: The average means of the observed probabilities for language combinations of length n and the expected probabilities based on language combinations of length $n-1$, $n-2$ and $n-3$ (where possible).

Nr. of languages	Threshold	Accuracy	Precision	Recall	F-score	Duration
2	0.112	0.850	0.846	0.857	0.850	00:04:09
3	0.059	0.817	0.810	0.829	0.819	00:31:14
4	0.037	0.777	0.793	0.760	0.773	14:09:05

Table 7.2: The average thresholds, performances in terms of accuracy, precision, recall and f-score, and the run-time of the cognate-recognition system for language pairs, triples and quadruples.

Unfortunately, since the run-time for the system exploded exponentially so badly, I was not able to evaluate the system for five languages. Given the run-times for the language pairs, language triples and language quadruples, the average run-time per line can be expressed as $T = 0.0287^n \cdot e^{1.06n^2}$ with n the number of languages. This results in an estimation of, on average, 2683.67 seconds per line for five languages. Given that it has to process 700 lines (300 in the training set, 200 in the development set and 200 in the test set), the total run time would be about 51 days.

Thus, for the actual compiling of the list of cognates, I was not able to use the combination of five languages. Instead of on the five languages, I therefore ran the system on the combination of four languages with the highest performance: English, Danish, German and Swedish. Because of the fact that Dutch was taken out of the data, all lines that had only one word in them or had become a double due to the removal of the Dutch words were removed, resulting in 192,655 lines, instead of the initially planned 318,651. I also ran the system on the language combination that had the best results overall, which was Danish-Swedish. Danish and Swedish was run on 47,862 lines, also because of lines that became doubles or came to have only one word in them due to the removal of English, German and Dutch words. A small excerpt of the extracted cognates can be found in appendix A. As can be seen there, the system is not flawless: there are some errors.

7.4 SVM

The SVM that was run on the same data to be able to compare results uses a (simple) C-support vector classification with a linear kernel, $\gamma = 0.8$ and a penalty parameter of 3. The input is the lines (i.e. tuples of cognates and tuples of non-cognates) as strings, thus containing tab characters to delimit the fields, on which a tf-idf vectorizer was applied on character n-grams to extract features.

Languages	Threshold	Accuracy	Precision	Recall	F-score	Duration
DA-DE	0.148	0.814	0.863	0.746	0.800	00:04:56
DA-NL	0.138	0.880	0.885	0.871	0.878	00:03:36
DA-SV	0.105	**0.923**	0.895	**0.968**	**0.925**	00:04:20
DE-NL	0.119	0.891	**0.925**	0.849	0.886	00:05:03
DE-SV	0.125	0.843	0.860	0.817	0.838	00:04:51
EN-DA	0.106	0.850	0.824	0.889	0.855	00:03:55
EN-DE	0.087	0.795	0.776	0.825	0.800	00:04:30
EN-NL	0.087	0.822	0.795	0.866	0.829	**00:03:15**
EN-SV	0.085	0.874	0.841	0.921	0.879	00:03:33
NL-SV	0.117	0.805	0.794	0.820	0.806	00:03:35
DA-DE-NL	0.059	0.799	0.770	0.848	0.807	00:33:44
DA-DE-SV	0.081	0.884	0.873	0.896	0.885	00:43:50
DA-NL-SV	0.067	0.818	0.798	0.848	0.822	00:29:51
DE-NL-SV	0.061	0.849	0.877	0.810	0.842	00:31:57
EN-DA-DE	0.046	0.779	0.747	0.840	0.791	00:37:53
EN-DA-NL	0.062	0.779	0.778	0.778	0.778	00:25:15
EN-DE-NL	0.055	0.814	0.821	0.800	0.810	00:25:46
EN-DA-SV	0.057	0.785	0.765	0.823	0.793	00:31:44
EN-DE-SV	0.045	0.835	0.812	0.873	0.841	00:27:50
EN-NL-SV	0.059	0.827	0.863	0.778	0.818	00:24:28
DA-DE-NL-SV	0.052	0.814	0.824	0.798	0.811	18:59:14
EN-DA-DE-NL	0.031	0.770	0.758	0.791	0.774	14:01:41
EN-DA-DE-SV	0.032	0.816	0.811	0.820	0.816	15:28:06
EN-DA-NL-SV	0.040	0.781	0.882	0.645	0.745	07:26:49
EN-DE-NL-SV	0.031	0.706	0.691	0.744	0.717	14:49:33

Table 7.3: The thresholds, performances in terms of accuracy, precision, recall and f-score, and the run-time of the cognate-recognition system for all language combinations. The best results are in boldface.

The SVM was evaluated using 5-fold cross-validation. The results with the SVM were better than those with my system, and it was much faster and therefore able to evaluate five language. On average, the results did improve when given more languages. The average f-score for two languages was 0.854, and 0.879, 0.874 and 0.880 for three, four and five languages respectively. On the other hand, it was not able to produce rules or calculate observed and expected probabilities, whereas my system is. See for all performance measures for all combinations Table 7.4 on page 35. Ciobanu and Dinu's (2014) SVM, applied to language pairs of Romanian with Italian, French, Spanish and Portuguese, had an average f-score of 0.825. Their SVM used aligned word pairs as input and n-grams as features.

Languages	Accuracy	Precision	Recall	F-score
DA-DE	0.896	0.90	**0.90**	0.89
DA-NL	0.870	0.87	0.87	0.87
DA-SV	0.860	0.87	0.86	0.86
DE-NL	0.872	0.87	0.87	0.87
DE-SV	0.819	0.82	0.82	0.81
EN-DA	0.840	0.84	0.84	0.84
EN-DE	0.851	0.85	0.85	0.85
EN-NL	0.855	0.86	0.86	0.85
EN-SV	0.855	0.86	0.86	0.85
NL-SV	0.852	0.86	0.85	0.85
DA-DE-NL	**0.904**	0.90	**0.90**	**0.90**
DA-DE-SV	0.897	**0.91**	**0.90**	0.89
DA-NL-SV	0.877	0.89	0.88	0.87
DE-NL-SV	0.895	0.90	**0.90**	0.89
EN-DA-DE	0.872	0.87	0.87	0.87
EN-DA-NL	0.872	0.87	0.87	0.87
EN-DE-NL	0.897	0.90	**0.90**	**0.90**
EN-DA-SV	0.864	0.87	0.86	0.86
EN-DE-SV	0.882	0.88	0.88	0.88
EN-NL-SV	0.864	0.86	0.86	0.86
DA-DE-NL-SV	0.899	0.90	**0.90**	**0.90**
EN-DA-DE-NL	0.889	0.89	0.89	0.89
EN-DA-DE-SV	0.863	0.86	0.86	0.86
EN-DA-NL-SV	0.861	0.87	0.86	0.86
EN-DE-NL-SV	0.860	0.86	0.86	0.86
EN-DA-DE-NL-SV	0.884	0.88	0.88	0.88

Table 7.4: The performances in terms of accuracy, precision, recall and f-score for the SVM. The best results are in boldface.

8. Discussion

The observed and expected probabilities calculated by the system developed for this thesis suggest that, indeed, the more languages, the more confident the decision whether all words in a tuple are cognates is, which would confirm my hypothesis. However, my hypothesis seems to be disproven by the results of the cognate-recognition system as the performances decreased the more languages were used. On the other hand, the results of the SVM do confirm my hypothesis that using more languages in cognate recognition improves results.

The decrease of rule confidence (as reported in Figure 7.1) was, perhaps, to be expected: even though some rules may actually become more confident and more exact, such 'if Dutch has a <d> and German has a <t>, English has <d>', most rules become less exact, as more languages means more variation. This does mean, though, that language combinations of languages that are more closely related should give better results. This is indeed supported by the fact that the combination Danish-Swedish and German-Dutch bear the best results – those two pairs are the pairs of the most closely related languages. Adding another language will result in a combination of languages that are less closely related, resulting in lower results. My system can therefore also be used to measure linguistic distance between languages in future research.

The speed by which the rule confidences (Figure 7.1) decrease with more languages is also striking, especially when compared to the decrease of the probability thresholds (Table 7.2). The probability thresholds decrease less quickly than the average rule confidences, meaning that the distance between cognate tuples and non-cognate tuples decreases in terms of probability as the number of languages grows. This could perhaps be solved by slightly changing the way substring-transition-rule probabilities are calculated. The probabilities are now calculated as in Equation 8.1.

$$P_o(\lambda_1 : ... : \lambda_n) = \sqrt[n]{\frac{n(\lambda_1 \cap ... \cap \lambda_n)}{n(\lambda_1) \cdot ... \cdot n(\lambda_n)}} \tag{8.1}$$

Given that the relation between the average means of the substring-transition-rule probabilities and the number of languages approaches $\bar{P} = \frac{1}{n}$ (where $\bar{P}$ is the average probability and n is the number of languages; see Figure 7.1), this could perhaps be corrected by taking the $n^{2\text{th}}$ root instead of the n^{th} root, as in Equation 8.2.

$$P_o(\lambda_1 : ... : \lambda_n) = \sqrt[n^2]{\frac{n(\lambda_1 \cap ... \cap \lambda_n)}{n(\lambda_1) \cdot ... \cdot n(\lambda_n)}} \tag{8.2}$$

Doing this would result in the average means as in Table 8.1 on page 37. The average means are now much closer to each other. What kind of impact this has on the thresholds and other results is subject for future research, though.

Nr. of languages	Average mean
2	0.696
3	0.660
4	0.687
5	0.709

Table 8.1: The new average means of the found rules for the language combinations.

In the process of cognate extraction it was found that the list of cognate tuples found by the run on Danish and Swedish (as opposed to the run on English, Danish, German and Swedish) was larger. This is not unexpected. Of course, the combination Danish-Swedish had better results, but apart from that more cognates are to be found between Danish and Swedish given this list of possible cognates, as this list is extracted from a parallel corpus; all possible cognates are translations of each other, so all cognates the system will find are 'true' friends, as opposed to false friends. Since Danish and Swedish are so closely related (more so than with, say, German) they share more 'true' friends with each other than with German (Danish and Swedish are mutually intelligible; so much even that when talking to each other, Danes will speak Danish and Swedes will speak Swedish). The number of cognate quadruples (let alone quintuples) that are each other's translation are relatively rare.

Finding false-friend cognates could be done using cognate prediction or cognate production, in which, given a set of transition rules, the form of a possible cognate is predicted (Mulloni, 2007; Beinborn, Zesch, & Gurevych, 2013). My system's transition-rules learner could be very useful for this.

As for the run-times of the cognate-recognition system, it is a shame that they increased so rapidly that it was impossible to evaluate the combination of five languages. This has of course to do with the exponential and combinatorial approaches of the system, as explained in chapter 6.

It was also found that German severely impacted the run-time of the system. All runs that included German were on average longer than those without German. In the same way, all runs that included English or Dutch were on average faster (hence also that the run on the combination English-Dutch was the fastest). Danish and Swedish did not seem to have such an impact. This can be explained by the average length of the words. German has, on average, longer words than the other languages. This results in more ϵ-partitions the system has to consider, resulting in a longer run-time. Table 8.2 shows the average run-time of all runs with and without a specific language, as well as the difference.

Language	With L	Without L	Difference
DA	4:52:17	5:07:04	0:14:46
DE	5:29:20	2:39:27	-2:49:53
EN	4:29:43	6:32:49	2:03:06
NL	4:47:14	5:22:35	0:35:21
SV	4:55:32	4:52:11	-0:03:21

Table 8.2: The average run-times with and without specific languages. The final column shows the difference in run times. Notice that runs including German took, on average, almost three hours longer. Those with English took almost two hours shorter, which is partly due to English not writing compounds as one word and having relatively shorter words in general.

Given that the SVM has better results in general and its results tend to go up when given more languages, my system can be improved. To start with, the run-time can be and should be improved in order for the developed system to be more efficient and more usable. This can be done by finding a way that makes the system 'smarter' so that it reduces the number of possible combinations that have to be considered in alignment, taming the exponential blow-up.

Also changing the way substring-transition-rule probabilities are calculated should be looked into, as the average probabilities assigned to cognate pairs go down the more languages are processed, while they, ideally, should go up. It could perhaps also be interesting to train the system not on positive labels (cognates) only, but on negative labels (non-cognates) as well in order to discover which substring-transition rules are in fact useful. Maybe the transition rule `q:q` does not say anything about words being cognates. Possibly calculating information gain for every transition rule can be useful here.

The system's design in that it does not consider partitions of words in which parts contain both letters representing a consonant and letters representing a vowel may have to change as well. Even though it increases run-time, it does impair the system in its finding transition rules. For example, some consonants may only change after certain vowels. More specifically yet, Dutch <ou> corresponds with English <ol> before a <d> (so `old:oud`), which the current system cannot detect due to the fact that the system would need to allow for partitions with parts containing both letters representing a consonant and letters representing a vowel to find this rule.

The substring aligner can be improved in that it sometimes returns an alignment in which empty strings alternate, such as in Figure 8.1. This can be solved by disallowing the system to do so, just like Covington (1996) disallowed their system to have two 'skips' after one another.

^ch	ϵ	u	r	ch$
ϵ	^k	e	r	k$

Figure 8.1: A wrong substring alignment in which empty string alternate.

Substring alignment can also be improved by adding a feature so that it recognizes alignments that exhibit metathesis, such as in Figure 8.2. This also requires the allowing of partitions with parts containing both letters representing a consonant and letters representing a vowel.

^b	ur	n	$\epsilon$$
^b	ra	n	d$

Figure 8.2: A substring alignment with an alignment that exhibits metathesis.

By defining vowels, consonants and semivowels for every language separately, substring alignment could be improved as well. For instance, now it treats <y> as a semivowel for every language, while it can only act as a (semi-)consonant in English, and not in Danish, German, Dutch or Swedish.

Phonetic information could also help alignment on a substring level. If the system would prefer alignments between phonemes (or graphemes) that share the place of pronunciation or the manner of pronunciation, the results might improve. In a way, I already implemented some phonetic information by disallowing partitions with parts containing both letters representing a consonant and letters representing a vowel, and by assigning higher smoothed probabilities, so that vowels are more likely to transition into vowels and substrings are more likely to transition into themselves, but the system might definitely benefit from more profound phonetic information.

The current system was not designed to recognize cognates that are derived with different morphemes. For example, the Danish *anholdelse* 'arrest' and the Dutch *aanhouding* 'arrest' are cognates, but are derived using different morphemes (*-else* vs. *-ing*, which are not cognates, as far as morphemes can be cognates). In order for the system to recognize these two words as cognates, it would need to be able to do partial cognate recognition (List, Lopez, & Bapteste, 2016). However, in my attempt to improve the run-times I already added this feature, by having the system cut off the parts before and after the aligned vowel (see Figure 6.4).

The calculation of the probabilities of the substring transitions could also be improved by applying smoothing. Smoothing was already applied to transitions that were not attested in the training data, but not in those that were.

As for the actual cognate recognition (i.e. the assigning of the `Y` label or `N` label to tuples), there could be improvements as well, also given that the SVM works better. A back-off decision-making system could be used that starts looking for cognates in subsets of the tuple if it found that the whole tuple is not a tuple of cognates. In the current system, one could regret that, when even one of the five languages replaced the original cognate by a borrowing, the system will assign the `N` label to the tuple, dismissing the whole tuple, while the words of all other languages might be cognates. Therefore, if the system finds an `N` for the whole tuple, it should look for cognacy between words in subsets of the tuple. For example, to the tuple `and:og:und:en:och` it will assign (or at least, should) `N`, meaning that not all words in the tuple are cognates of each other. It should then check if the subsets `and:og:und:en`, `and:og:und:och`, `and:og:en:och`, `and:und:en:och` and `og:und:en:och` are cognates. It should, again, find that they are not. Then it should look in even smaller subsets, until it finds that `and:und:en` is a cognate tuple and `og:och` is as well. This back-off decision-making system should significantly improve results.

It may also be interesting to see how results are influenced if the system will use the transitivity of cognacy (if A is a cognate of B and B is a cognate of C, then A is a cognate of C). However, from what we have seen regarding expected and observed probabilities, using transitivity should make the decisions of cognacy less confident. Nonetheless, it might be useful.

The Europarl corpus might not have been the most suitable corpus for this research, especially given that the system was trained on (an extended version of) the Swadesh list. The Europarl corpus contains relatively much jargon, terminology, names and loans, and relatively few old 'normal' words. The transition rules that the system learned might only be relevant for old 'normal' words, and not for younger words which are not old enough to have undergone all the same sound changes. On the other hand, this might have the system distinguish better between loans and true cognates and maybe even calques.

Despite all these possible advantageous adjustments that can be made to the developed system, the results of this thesis place themselves rather well in the existing literature of cognate recognition. The accuracy, precision, recall and f-scores of all runs are comparable, sometimes even better, than other systems. The results can also be, as said, very useful for cognate prediction. It could, following from this, even be used for language reconstruction, because this system returns a list of transition rules it has found, which in turn could be studied thoroughly. It would be interesting to compare the found transition rules with those the comparative grammar of Germanic languages holds for sound (Plotkin, 2008).

Some researchers have suggested using transition rules in string similarity measures (Nakov, Nakov, & Paskaleva, 2009). The rules this system returns can be used for this purpose excellently.

Because of its universal nature, this system cannot only be run on the five languages focused on in this thesis, but can be run on all combinations of languages, even regardless of the alphabet they are written in. Not only that, but the system can also be run on morphological data. Whereas Albright and Hayes's (2006) MGL can be used to discover patterns of vowel changes between present tense and past tense in Germanic strong verbs, my system can be used to discover patterns of vowel changes between present tense, past tense and past participles at the same time. In a way, I present a multilateral MGL that even allows for the aligning of substrings of different lengths, which the original MGL cannot do.

Furthermore it may be interesting to see how the adding of syntactic information to words would influence the results of cognate recognition. Of course nouns are more likely to be cognates with other nouns (strictly speaking, it is even so that only nouns can be cognates with other nouns). Adding POS tags to words might therefore be beneficial and might result in better cognate recognition.

It might also be interesting to see whether the use of diachronic corpora could benefit cognate recognition. As said, the transition rules found in the (extended) Swadesh list might only apply to old words. When given a diachronic corpus, the system might be able to find other rules for words of different ages, and as a result of that be able to distinguish between true cognates and borrowings. It would also, then, make fewer errors regarding non-cognate pairs that share huge resemblance by chance, such as Latin *dies* and English *day* (Harper, 2016).

Another approach that could be beneficial is when the system adjusts its transition rules every time it finds a new cognate tuple: some sort of bootstrapping. If, for instance, the system newly finds that the tuple `house:hus:Haus:huis:hus` is a cognate tuple, it should change all transition rules with `h`s, `ou`s, `s`s, `e`s, etc. so that they are adjusted for the newly found transitions. Whether this would improve the results, I am not sure. I expect it would lead to very inexact rules, because it applies some sort of training on errors as well, leading to erroneous transition rules, which will in turn lead to more errors. This requires further looking into, though.

In short, the developed system could use many improvements. Nonetheless, the results are not bad at all compared to already existing systems. The system presented here also fills a gap in the literature in that it is a multilingual cognate-recognition system which also returns multilateral transition rules. These multilateral transition rules have also proven to be useful, as they result in higher probabilities than would be expected when combining transition rules with fewer languages. Especially given the results of the SVM, I have shown that cognate recognition benefits from adding languages in the equation. All improvements of the system and applications of the results, however, I leave for future research.

Bibliography

Albright, A. & Hayes, B. (2006). Modeling productivity with the gradual learning algorithm: the problem of accidentally exceptionless generalizations. *Gradience in grammar: Generative perspectives*, 185–204.

Barker, G. & Sutcliffe, R. F. (2000). An experiment in the semi-automatic identification of false-cognates between English and Polish. In *Proceedings of the Irish conference on artificial intelligence and cognitive science*.

Beinborn, L., Zesch, T., & Gurevych, I. (2013). Cognate production using character-based machine translation. In *Proceedings of the 6th international joint conference on natural language processing* (pp. 883–891).

Bergsma, S. & Kondrak, G. (2007a). Alignment-based discriminative string similarity. In *Proceedings of the 45th annual meeting of the ACL* (pp. 656–663).

Bergsma, S. & Kondrak, G. (2007b). Multilingual cognate identification using integer linear programming. In *RANLP workshop on acquisition and management of multilingual lexicons*.

Borin, L., Forsberg, M., & Lönngren, L. (2013). SALDO: a touch of yin to WordNet's yang. *Language resources and evaluation*, *47*(4), 1191–1211.

Brew, C., McKelvie, D. et al. (1996). Word-pair extraction for lexicography. In *Proceedings of the 2nd international conference on new methods in language processing* (pp. 45–55).

Ciobanu, A. M. & Dinu, L. P. (2014). Automatic detection of cognates using orthographic alignment. In *Proceedings of the 52nd annual meeting of the Association for Computational Linguistics* (Vol. 2, pp. 99–105).

Covington, M. A. (1996). An algorithm to align words for historical comparison. *Computational linguistics*, *22*(4), 481–496.

Cysouw, M. & Jung, H. (2007). Cognate identification and alignment using practical orthographies. In *Proceedings of the 9th meeting of the ACL: special interest group in computational morphology and phonology* (pp. 109–116). Association for Computational Linguistics.

Danielsson, P. & Muehlenbock, K. (2000). Small but efficient: the misconception of high-frequency words in Scandinavian translation. In *Conference of the Association for Machine Translation in the Americas* (pp. 158–168). Springer.

Dyer, C., Lopez, A., Ganitkevitch, J., Weese, J., Ture, F., Blunsom, P., … Resnik, P. (2010). cdec: a decoder, alignment, and learning framework for finite-state and context-free translation models. In *Proceedings of the 48th annual meeting of the Association for Computational Linguistics*.

Frunza, O. & Inkpen, D. (2006). Semi-supervised learning of partial cognates using bilingual bootstrapping. In *Proceedings of the 21st international conference on computational linguistics and the 44th annual meeting of the Association for Computational Linguistics* (pp. 441–448). Association for Computational Linguistics.

Gomes, L. & Lopes, J. G. P. (2011). Measuring spelling similarity for cognate identification. In *Portuguese conference on artificial intelligence* (pp. 624–633). Springer.

Guy, J. B. M. (1994). An algorithm for identifying cognates in bilingual word-lists and its applicability to machine translation. *Journal of Quantitative Linguistics*, *1*(1), 35–42.

Hall, D. & Klein, D. (2010). Finding cognate groups using phylogenies. In *Proceedings of the 48th annual meeting of the Association for Computational Linguistics* (pp. 1030–1039). Association for Computational Linguistics.

Hall, D. & Klein, D. (2011). Large-scale cognate recovery. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 344–354). Association for Computational Linguistics.

Harper, D. (2016). Online etymology dictionary. Retrieved August 21, 2016, from http://www.etymonline.com/

Inkpen, D., Frunza, O., & Kondrak, G. (2005). Automatic identification of cognates and false friends in french and english. In *RANLP* (pp. 251–257).

Koehn, P. & Knight, K. (2000). Estimating word translation probabilities from unrelated monolingual corpora using the EM algorithm. In *AAAI/IAAI* (pp. 711–715).

Kondrak, G. (2001). Identifying cognates by phonetic and semantic similarity. In *Proceedings of the 2nd meeting of the North American chapter of the Association for Computational Linguistics on language technologies* (pp. 1–8). Association for Computational Linguistics.

Kondrak, G. (2004). Combining evidence in cognate identification. In *Conference of the Canadian Society for Computational Studies of Intelligence* (pp. 44–59). Springer.

Kondrak, G., Marcu, D., & Knight, K. (2003). Cognates can improve statistical translation models. In *Proceedings of the 2003 conference of the North American chapter of the Association for Computational Linguistics on human language technology: companion volume of the proceedings of HLT-NAACL* (Vol. 2, pp. 46–48). Association for Computational Linguistics.

Kromann, M. & Lynge, S. (2004). Danish Dependency Treebank v. 1.0. Department of Computational Linguistics, Copenhagen Business School.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, *10*(8), 707–710.

Lewis, M. P., Simons, G. F., & Fennig, C. D. (Eds.). (2016). *Ethnologue: Languages of the world* (19th ed.). Dallas, Texas: SIL International. Retrieved August 19, 2016, from http://www.ethnologue.com

List, J.-M. (2012). LexStat: automatic detection of cognates in multilingual wordlists. In *Proceedings of the EACL 2012 joint workshop of LINGVIS & UNCLH* (pp. 117–125). Association for Computational Linguistics.

List, J.-M., Lopez, P., & Bapteste, E. (2016). Using sequence similarity networks to identify partial cognates in multilingual wordlists. In *Proceedings of the 54th annual meeting of the Association for Computational Linguistics* (pp. 599–605). Association for Computational Linguistics.

Malmasi, S., Dras, M. et al. (2015). Cognate identification using machine translation. In *Australasian language technology association workshop 2015* (p. 138).

Mann, G. S. & Yarowsky, D. (2001). Multipath translation lexicon induction via bridge languages. In *Proceedings of the 2nd meeting of the North American chapter of the Association for Computational Linguistics on language technologies* (pp. 1–8). Association for Computational Linguistics.

McColl Millar, R. & Trask, R. L. (2007). *Trask's historical linguistics* (2nd ed.). London: Hodder Arnold.

Melamed, D. I. (1995). Automatic evaluation and uniform filter cascades for inducing n-best translation lexicons. In *Proceedings of the 3rd workshop on very large corpora.*

Mulloni, A. (2007). Automatic prediction of cognate orthography using support vector machines. In *Proceedings of the 45^{th} annual meeting of the ACL: student research workshop* (pp. 25–30). Association for Computational Linguistics.

Mulloni, A. & Pekar, V. (2006). Automatic detection of orthographic cues for cognate recognition. In *Proceedings of LREC'06* (pp. 2387–2390).

Nakov, S., Nakov, P., & Paskaleva, E. (2009). Unsupervised extraction of false friends from parallel bi-texts using the web as a corpus. In *RANLP* (pp. 292–298).

Onions, C. T. (1966). *The Oxford dictionary of English etymology*. Oxford: Clarendon.

jalu.ch – One letter words. (2016). Retrieved August 19, 2016, from http://jalu.ch/languages/one_letter_words.php

Pagel, V., Lenzo, K., & Black, A. W. (1998). Letter-to-sound rules for accented lexicon compression. In *Proceedings of the international conference on spoken language processing* (Vol. 5, pp. 2015–2018). Sydney, Australia.

Plotkin, V. (2008). *The evolution of Germanic phonological systems: Proto-Germanic, Gothic, West Germanic, and Scandinavian*. Edwin Mellen Press.

Rama, T. (2015). Automatic cognate identification with gap-weighted string subsequences. In *Proceedings of the 2015 conference of the North American chapter of the Association for Computational Linguistics: human language technologies* (pp. 1227–1231). Denver, Colorado, USA.

Simard, M., Foster, G. F., & Isabelle, P. (1993). Using cognates to align sentences in bilingual corpora. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: distributed computing* (Vol. 2, pp. 1071–1082). IBM Press.

Swadesh, M. (1955). Towards greater accuracy in lexicostatistic dating. *International Journal of American linguistics*, *21*(2), 121–137.

Swedish Institute et al. (2015). Learning Swedish. Retrieved May 16, 2016, from http://learningswedish.se/

Tiedemann, J. (2012). Parallel data, tools and interfaces in opus. In *Proceedings of LREC'12* (pp. 2214–2218).

Trask, R. L. (2000). *The dictionary of historical and comparative linguistics*. Edinburgh: Edinburgh University Press.

Wang, H. & Sitbon, L. (2014). Multilingual lexical resources to detect cognates in non-aligned texts. In *Proceedings of the Australasian Language Technology Association Workshop 2014* (Vol. 12, pp. 14–22).

Winkler, W. E. (1990). String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. In *Proceedings of the Section on Survey Research Methods* (pp. 354–359). American Statistical Association.

Xu, Q., Chen, A., & Li, C. (2015). Detecting English-French cognates using orthographic edit distance. In *Proceedings of Australasian Language Technology Association Workshop* (pp. 145–149).

A. Excerpt of database

A.1 English, Danish, German and Swedish

EN	DA	DE	SV
senegalese	senegalesere	senegalese	senegales
pressure			pressa
practice		umsetzung	
jalla	jalla	jalla	jalla
bury			begrava
site		website	
democracy	demokratifond		demokratifond
commissioner			kommissionär
territorial	territoriale		territorium
bus			bus
project			projekt
guarantee			garantera
heterosexual	heteroseksuel	heterosexuell	heterosexuella
routine			rutin
provide		erstellen	
usa		usa	
materialism	materialisme	materialismus	materialism
uncontrolled		unkontrolliert	okontrollera
have			avlös
robinson	robinson		robinson
instead		anstatt	
have		sehen	
regionalisation	regionalisering	regionalisierung	
wil	således		
finlandisation	finlandisering	finlandisierung	finlandisering
perpetually		ständig	
bi-regional	biregional	biregionalen	biregional
security	security		
olle			olle
introduce		einbringen	
financial	finansverdene	finanzwelt	finansvärld
formulate		formulierung	formulera
concern		sorgen	
millennium	millennium	millennium	millennium
green	grøne		grön
roulette	roulette	roulette	roulette

Continued

EN	DA	DE	SV
reconstruction	genopbygning		
africa			östafrika
energy-efficient	energieffektiv		energieffektiv
de	de	sehr	
hinge	afhænge		
assistant	assistente		assistent
standard		standard	standard
combined	kombinere		
salman	salman	salman	salman
murko	murko		murko
concrete	konkret		konkret
report	report	report	report
fish		fisch	
anomaly	anomali	anomalie	anomali
coleague	kolega	kolegin	
gas		gas	
seminar		seminar	seminarium
den			den
shall		lassen	
where		geraten	
development	development		
retain		erhalten	
lira	lire	lira	lira
isolation			isolering
fly			flyga
bernardino	bernardino	bernardino	bernardino
record		werden	
call	kalde		
supplementary		zusätzlich	
financial			finanskris
classical	klassisk	klassisch	klassisk
norbert	norbert	norbert	norbert
atlantic		atlantisch	
economics	økonomi		ekonomisk
kallas		kallas	kallas
rule		regel	
oli		oli	oli
continental	continental	continental	continental
talent	talent		
on	en		en
integrity	integritet	integrität	
an	en		
provide	angive		
around		ansetzen	

A.2 Danish and Swedish

DA	SV
nordirland	nordirland
attali	attali
bono	bono
inddeling	indelning
santer	santer
overalt	överall
milinkievitj	milinkevitj
medvirke	medverka
ironi	ironisk
mongoler	mongol
programs	program
forbundsråd	förbundsråd
justeres	justera
ikke-diskriminering	ickediskriminering
kemikalierne	kemikalie
fly	flyg
udvælge	utvald
placering	placering
konsekvens	konsekvent
genemføre	genomföras
patakis	patakis
institutionalisering	institutionalisera
populistiske	populistiskt
vertikal	vertikal
eksportør	exportör
stilistisk	stilistisk
elev	elev
lade	laden
norm	norm
interpol	interpol
andry	andry
djakourmas	tsiakourmas
underskrift	underskrift
segni	segni
indefra	inifrån
sis	sis
undersøge	undersökning
digitalt	digital-tv
synder	syndare
kernekraft	kärnkraften
koordineret	koordinerad

Continued

DA	SV
ats	ats
ineffektivitet	ineffektivitet
helte	hjälte
absurdum	absurdum
skotland	skottland
vise	viser
certificering	certifiering
over	går
gensidig	ömsesidiga
permanent	permanent
balkan	balkan
velinformerede	välinformera
jadot	jadot
økonomi	ekonomisk
reduktion	reduktion
funk	funk
lærling	lärlingar
åbning	öpning
sabine	sabine
gestapo	gestapo
såsom	såsom
fastsætte	fastställa
legitimitet	legitimt
fodfæste	fotfäste
institut	institute
rekrutering	nyrekrytering
terroriserede	terrorisera
overtrædelse	överträdelse
regulatory	regulatory
endesa	endesa
medine	medine
prostitueret	prostituera
ods	ods
forvaltningsret	förvaltningsrät
dogmatismen	dogmatism
uforudsete	oförutsed
introducere	introducera
velkomment	välkommet
kolegialt	kolegial

B. Excerpt of found transition rules

EN	DA	DE	NL	SV	P
rn$	rn$	rn$	rn$	rn$	1.000
ls	ls	ls	ls	ls	1.000
^kn	^kn	^kn	^kn	^kn	1.000
ly$	lg$	lg$	lg$	lg$	1.000
rm$	rm$	rm$	rm$	rm$	1.000
^gl	^gl	^gl	^gl	^gl	1.000
^sn	^sn	^schn	^sn	^sn	1.000
^br	^br	^br	^br	^br	1.000
mp$	mp$	mpf$	mp$	mp$	1.000
^bl	^bl	^bl	^bl	^bl	1.000
^sl	^sl	^schl	^sl	^sl	0.851
sh$	sk$	sch$	s$	sk$	0.833
^r	^r	^r	^r	^r	0.787
^m	^m	^m	^m	^m	0.748
x$	ks$	chs$	s$	x$	0.725
^u	^o	^u	^o	^o	0.723
^thr	^tr	^dr	^dr	^tr	0.712
mb$	m$	m$	m$	m$	0.699
ght$	t$	cht$	cht$	t$	0.696
^b	^b	^b	^b	^b	0.684
^sp	^sp	^sp	^sp	^sp	0.683
ft$	ft$	ft$	cht$	ft$	0.678
^gr	^gr	^gr	^gr	^gr	0.665
^ey	^æ	^ei	^ei	^ä	0.660
^fl	^fl	^fl	^vl	^fl	0.629
ck$	k$	ck$	k$	ck$	0.625
^st	^st	^st	^st	^st	0.624
lt$	lt$	lz$	t$	lt$	0.619
^h	^h	^h	^h	^h	0.613
ee$	æ$	ie$	ie$	ä$	0.608
^f	^f	^f	^v	^f	0.601
nd$	nd$	nd$	nd$	nd$	0.597
^tw	^t	^zw	^tw	^tv	0.582
s$	s$	s$	s$	s$	0.582
^l	^l	^l	^l	^l	0.576
ry$	rg$	rg$	rg$	rg$	0.574
^str	^str	^str	^str	^str	0.574
o$	o$	ei$	ee$	å$	0.574

Continued

EN	DA	DE	NL	SV	P
rt$	rt$	rz$	rt$	rt$	0.574
ng$	ng$	ng$	ng$	ng$	0.561
r$	r$	r$	r$	r$	0.558
^a	^a	^a	^a	^a	0.517
p	b	f	p	p	0.506
^ea	^æ	^e	^e	^ä	0.500
^ϵ	^j	^ϵ	^ϵ	^j	0.500
^sw	^s	^schw	^zw	^sv	0.495
f$	v$	b$	f$	v$	0.488
^o	^e	^ei	^ee	^e	0.488
^fr	^fr	^fr	^vr	^fr	0.484
^g	^g	^g	^g	^g	0.467
ea$	ø$	ee$	ee$	ö$	0.461
^w	^v	^w	^w	^v	0.460
^t	^t	^z	^t	^t	0.445
^dr	^dr	^tr	^dr	^dr	0.444
^sh	^sk	^sch	^sch	^sk	0.435
^n	^n	^n	^n	^n	0.432
sh	sk	sch	s	sk	0.427
^e	^e	^e	^e	^e	0.425
^ea	^ø	^o	^oo	^ö	0.401
^e	^o	^au	^oo	^o	0.392
^fl	^fl	^fl	^fl	^fl	0.392
ow$	rg$	rg$	rg$	rg$	0.379
gh	gt	cht	cht	kt	0.370
p	p	p	u	p	0.367
d$	d$	t$	d$	d$	0.367
e$	$\epsilon$$	$\epsilon$$	$\epsilon$$	$\epsilon$$	0.366
oy	øj	eu	ui	y	0.361
^th	^d	^d	^d	^d	0.361
r	rn	rn	r	rn	0.354
w	lm	we	we	ä	0.347
tc	k	ck	k	ck	0.343
sk	sk	sh	s	sk	0.339
v	v	b	v	v	0.335
rth$	rt$	rz$	rt$	rt$	0.330
^d	^dr	^tr	^dr	^d	0.329
^st	^stj	^st	^st	^stj	0.325
m	vn	m	m	mn	0.310
ld$	l$	ld$	d$	l$	0.301
l$	l$	l$	l$	l$	0.298
^s	^s	^s	^z	^s	0.297
ng	ng	ng	ng	ng	0.295
t$	gt$	cht$	cht$	kt$	0.293

Continued

EN	DA	DE	NL	SV	P
oo	o	u	oe	o	0.293
^f	^f	^v	^v	^f	0.292
^k	^kv	^q	^kw	^kv	0.291
	g	g	g	g	0.290
ew	y	eie	uw	y	0.288
d	d	t	d	d	0.287
s	s	s	s	s	0.273
^k	^k	^k	^k	^k	0.268
a	a	a	a	a	0.264
ϵ$	ϵ$	n$	n$	ϵ$	0.260
v	m	nf	jf	m	0.257
n	k	nk	nk	ck	0.241
ld$	ld$	ld$	d$	l$	0.239
r	r	r	r	r	0.238
tch$	g$	ch$	k$	k$	0.237
n	n	n	n	n	0.236
^b	^bj	^b	^b	^b	0.231
th$	d$	r$	r$	d$	0.225
ckl	gl	ch	k	g	0.224
z	s	r	z	s	0.224
w	∅	wi	we	i	0.223
k$	g$	ch$	k$	k$	0.218
mb	m	m	m	m	0.217
th$	nd$	hn$	nd$	nd$	0.214
n	ng	ng	ng	ng	0.211
ee	y	ie	ie	y	0.204
^w	^ϵ	^w	^w	^ϵ	0.196
r	r	rz	rt	r	0.195
ow$	e$	ee$	eeuw$	ö$	0.195
ee$	e$	ei$	ie$	e$	0.195
l	l	l	l	l	0.192
t	d		t	t	0.192
ck$	g$	ch$	k$	ck$	0.190
l	ld	lt	d	l	0.189
^dr	^dr	^tr	^tr	^dr	0.188
b	b	f	f	p	0.187
^d	^dr	^tr	^dr	^dr	0.186
s	s	s	z	s	0.184

C. Extended Swadesh list

EN	DA	DE	NL	SV
I	jeg	ich	ik	jag
you	jer	euch	jou	er
you	jer	euch	u	er
ye	i	ihr	jij	i
ye	i	ihr	gij	i
thou	du	du		du
he			hij	
he			ie	
	han			han
we	vi	wir	wij	vi
they	de			de
		sie	zij	
this	dette	dies	dit	detta
that	det	das	dat	det
that	det	dass	dat	det
that	det	daß	dat	det
here	her	hier	hier	här
there	der	da	daar	där
who	hvem	wer	wie	vem
what	hvad	was	wat	vad
where	hvor	wo	waar	var
when		wenn	wen	
when		wann	wen	
	hvornår		wanneer	
	når			när
how		wie	hoe	hur
not		nicht	niet	
	ikke			icke
all	al	all	al	all
many	mangen	manch	menig	mången
fele		viel	veel	
some	somme	summig	sommig	
any		einig	enig	
	nogen			någon
few	få			få
		wenig	weinig	
other	anden	ander	ander	annan
one	en	ein	een	en

Continued

EN	DA	DE	NL	SV
two	to	zwei	twee	två
three	tre	drei	drie	tre
four	fire	vier	vier	fyra
five	fem	fünf	vijf	fem
stoor	stor	stur	stoer	stor
great		groß	groot	
long	lang	lang	lang	lång
wide	vid	weit	wijd	vid
broad	bred	breit	breed	bred
thick	tyk	dick	dik	tjock
heavy		hebig	hevig	
sweer		schwer	zwaar	
	tung			tung
small	små	schmal	smal	små
little	liden	lützel	luttel	liten
clean		klein	klein	
short	skort			
	kort	kurz	kort	kort
narrow		Narbe	naar	
		eng	eng	
throng	trang	Drang	drang	trång
thin	tynd	dünn	dun	tunn
queen	kvinde		kween	kvinna
queen	kone		kween	kvinna
queen	kvinde		kween	kona
queen	kone		kween	kona
	frue	Frau	vrouw	fru
	fru	Frau	vrouw	fru
man	mand	Mann	man	man
man	mand	man	man	man
man	mand	Mann	men	man
man	mand	man	men	man
churl	karl	Kerl	kerel	karl
	menneske	Mensch	mens	människa
child	kuld			kull
kind		Kind	kind	
barn	barn			barn
wife	viv	Weib	wijf	viv
		Gatte	gade	
	hustru			hustru
husband	husbonde			husbonde
	mage	Macker	makker	make
mother	mor	Mutter	moeder	mor
father	far	Vater	vader	far

Continued

EN	DA	DE	NL	SV
mother	moder	Mutter	moeder	mor
father	fader	Vater	vader	far
beast		Bestie	beest	best
deer	dyr	Tier	dier	djur
fish	fisk	Fisch	vis	fisk
fowl	fugl	Vogel	vogel	fågel
hound	hund	Hund	hond	hund
louse	lus	Laus	luis	lus
snake	snog	Schnake	snaak	snok
	slange	Schlange	slang	
wyrm	orm	Wurm	worm	orm
worm	orm	Wurm	worm	orm
maddock	maddike	Made	made	matk
tree	træ		teer	träd
tree	træ		teer	trä
beam	bom	Baum	boom	
wold	vold	Wald	woud	vall
weald	vold	Wald	woud	vall
weld	vold	Wald	woud	vall
wold	val	Wald	woud	vall
weald	val	Wald	woud	vall
weld	val	Wald	woud	vall
bush	busk	Busch	bos	buske
scough	skov			skog
stock	stok	Stock	stok	stock
stick	stikke			sticka
pin	pind			pinne
stave	stav	Stab	staf	stav
staff	stav	Stab	staf	stav
stave	stav	Stab	staaf	stav
staff	stav	Stab	staaf	stav
fruit	frugt	Frucht	vrucht	frukt
ovest		Obst	ooft	
seed	sæd	Saat	zaad	säd
fry	frø			frö
leaf	løv	Laub	loof	löv
blade	blad	Blatt	blad	blad
wort	urt	Wurzel	wortel	ört
root	rod			rot
bark	bark		bark	bark
bloom	blomme	Blume	bloem	blomma
grass	græs	Gras	gras	gräs
rope	reb	Reif	reep	rep
rope	reb	Reif	roop	rep

Continued

EN	DA	DE	NL	SV
tow		Tau	touw	
tie	tov			tåg
		Seil	zeel	
hide	hud	Haut	huid	hud
skin	skind	Schinde	schinde	skinn
fell	fjeld	Fell	vel	fjäll
flesh	flæsk	Fleisch	vlees	fläsk
meat	mad		met	mat
	kød			kött
blood	blod	Blut	bloed	blod
bone	ben	Bein	been	ben
knuckle	knogle	Knochen	knokkel	knoge
fat	fedt	Fett	vet	fett
egg	æg	Ei	ei	ägg
ey	æg	Ei	ei	ägg
horn	horn	Horn	hoorn	horn
tail	tavl	Zagel	teil	tagel
start	stjært	Sterz	staart	stjärt
	svans	Schwanz		svans
feather	fjer	Feder	veer	fjäder
feather	fjer	Feder	veder	fjäder
hair	hår	Haar	haar	hår
head	hoved	Haupt	hoofd	huvud
cup		Kopf	kop	
ear	øre	Ohr	oor	öra
eye	øje	Auge	oog	öga
nose	næse	Nase	neus	näsa
nose	næse	Nase	neus	nos
mouth	mund	Mund	mond	mun
mouth	mund	Mund	muide	mun
		Maul	muil	
tooth	tand	Zahn	tand	tand
tongue	tunge	Zunge	tong	tunga
nail	negl	Nagel	nagel	nagel
finger	finger	Finger	vinger	finger
foot	fod	Fuß	voet	fot
leg	læg			lägg
knee	knæ	Knie	knie	knä
hand	hånd	Hand	hand	hand
wing	vinge			vinge
		Flügel	vleugel	
maw	mave	Magen	maag	mage
bouk	bug	Bauch	buik	buk
bellow	bælg	Balg	balg	bälg

Continued

EN	DA	DE	NL	SV
belly	bælg	Balg	balg	bälg
yote	gyde	gießen	gieten	gjuta
tharm	tarm	Darm	darm	tarm
neck	nakke	Nacken	nek	nacke
halse	hals	Hals	hals	hals
ridge	ryg	Rücken	rug	rygg
back	bag		bak	bak
breast	bryst	Brust	borst	bröst
heart	hjerte	Herz	hart	hjärta
liver	lever	Leber	lever	lever
drink	drikke	trinken	drinken	dricka
eat	æde	essen	eten	äta
	spise	Speise	spijs	spisa
bite	bide	beissen	bijten	bita
suck	suge	saugen	zuigen	suga
spew	spy	speien	spuwen	spy
spew	spy	speien	spugen	spy
spit	spid	Spieß	spit	spett
blow	blæse	blasen	blazen	blåsa
	vaje	wehen	waaien	vaja
breathe		Brodem	bradem	
		atmen	ademen	
	ånde	ahnden		andas
laugh	le	lachen	lachen	le
	grine	greinen	grienen	grina
	grine	greinen	grijnen	grina
	skratte			skratta
see	se	sehen	zien	se
hear	høre	hören	horen	höra
wit	vide	wissen	weten	veta
wit	vide	wissen	weten	vita
ken	kende	kennen	kennen	känna
think	tænke	denken	denken	tänka
reek	ryge	riechen	ruiken	ryka
reek	ryge	riechen	rieken	ryka
smell	smul		smeulen	
	lugte			lukta
fear	fare	Gefahr	gevaar	fara
sleep		schlafen	slapen	
sweb	sove			sova
swab	sove			sova
live	leve	leben	leven	leva
die	dø			dö
starve		sterben	sterven	

Continued

EN	DA	DE	NL	SV
kill	kvæle	quälen	kwellen	kvälja
quell	kvæle	quälen	kwellen	kvälja
	døde	töten	doden	döda
		umbringen	ombrengen	
drub	dræbe	treffen	treffen	dräpa
drib	dræbe	treffen	treffen	dräpa
fight	fægte	fechten	vechten	fäkta
fight	fegte	fechten	vechten	fäkta
stride		streiten	strijden	strida
	kæmpe	kämpfen	kampen	kämpa
	kæmpe	kämpfen	kempen	kämpa
	slås			slåss
	jage	jagen	jagen	jaga
slay	slå	schlagen	slagen	slå
slay	slå	schlagen	slaan	slå
hit	hitte			hitta
cut				kuta
cut				kåta
snithe	snide	schneiden	snijden	snida
shear	skære	scheren	scheren	skära
split	splitte	spleißen	splijten	
cleave	kløve	klieben	klieven	klyva
deal	dele	teilen	delen	dela
	skille			skilja
stick	stikke	stechen	steken	sticka
cratch	kradse	kratzen	kratse	kratsa
grave	grave	graben	graven	gräva
grave	grave	graben	graven	grava
dig	dige			dika
delve		telben	delven	
delve		delben	delven	
swim	svømme	schwimmen	zwemmen	simma
fly	flyve	fliegen	vliegen	flyga
walk	valke	walken	walken	
waulk	valke	walken	walken	
leap	løbe	laufen	lopen	löpa
step			stappen	
go	gå	gehen	gaan	gå
come	komme	kommen	komen	komma
lie	ligge	liegen	liggen	ligga
sit	sidde	sitzen	zitten	sitta
stand	stå	stehen	staan	stå
rise	rejse	reisen	rijzen	risa
throw	dreje	drehen	draaien	dreja

Continued

EN	DA	DE	NL	SV
wend	vende	wenden	wenden	vända
fall	falde	fallen	vallen	falla
give	give	geben	geven	ge
give	give	geben	geven	giva
yive	give	geben	geven	ge
yive	give	geben	geven	giva
hold	holde	halten	houden	hålla
squeeze		quetschen	kwetsen	kväsa
	klemme		klemmen	klämma
		kneifen	knijpen	
thrutch	trykke	drucken	drukken	trycka
thrutch	trykke	drücken	drukken	trycka
rub	rubbe			
		reiben	wrijven	
	gnide			gnida
wash	vaske	waschen	wassen	vaska
	tvætte			tvätta
wipe		wippen		veva
	viske	wischen	wissen	viska
rinse	rense			rensa
drag	drage	tragen	dragen	draga
drag	drage	tragen	dragen	dra
drag	drage	tragen	dragen	dragga
drag	drægge	tragen	dragen	draga
drag	drægge	tragen	dragen	dra
drag	drægge	tragen	dragen	dragga
draw	drage	tragen	dragen	draga
draw	drage	tragen	dragen	dra
draw	drage	tragen	dragen	dragga
draw	drægge	tragen	dragen	draga
draw	drægge	tragen	dragen	dra
draw	drægge	tragen	dragen	dragga
	trække	trechen	trekken	
tee		ziehen	tijgen	
shove	skubbe	schieben	schuiven	skjuva
warp	værpe	werfen	werpen	värpa
cast	kaste			kasta
bind	binde	binden	binden	binda
sew	sy			sy
		nähen	naaien	
tell	tælle	zählen	tellen	tälja
reckon	regne	rechnen	rekenen	räkna
say	sige	sagen	zeggen	säga
sing	synge	singen	zingen	sjunga

Continued

EN	DA	DE	NL	SV
plaw	pleje	pflegen	plegen	pläga
plaw	pleje	pflegen	plegen	pläga
speel	spille	spielen	spelen	spela
lake	lege			leka
fleet	flyde	fließen	vlieten	flyta
drive	drive	treiben	drijven	driva
		schweben	zweven	
glide	glide	gleiten	glijden	glida
flow			vloeien	
stream	strømme	strömen	stromen	strömma
rin	rinde	rennen	rennen	rinna
rin	rinde	rinnen	rennen	rinna
run	rinde	rennen	rennen	rinna
run	rinde	rinnen	rennen	rinna
freeze	fryse	frieren	vriezen	frysa
swell	svulme	schwellen	zwellen	svälla
sun		Sonne	zon	
	sol			sol
moon	måne	Mond	maan	måne
star	stjerne	Stern	ster	stjärna
water	vand	Wasser	water	vatten
rain	regne	Regen	regen	regn
river		Revier	rivier	
flood	flod	Flut	vloed	flod
	elv	Elbe		älv
lake		Lache	laak	
sea	sø	See	zee	sjö
mere	mar	Meer	meer	mar
	hav	Haff		hav
salt	salt	Salz	zout	salt
stone	sten	Stein	steen	sten
sand	sand	Sand	zand	sand
dust	dyst	Dust	duist	dust
	støv	Staub	stof	
earth	jord	Erde	aarde	jord
bottom	bund	Boden	bodem	botten
cloud	klode	Kloß	kluit	klot
welkin		Wolke	wolk	
sky	sky			sky
mist	mist		mist	mist
		Nebel	nevel	
	tåge			töcken
	tåge			tjocka
	himmel	Himmel	hemel	himmel

Continued

EN	DA	DE	NL	SV
lift	luft	Luft	lucht	luft
wind	vind	Wind	wind	vind
snow	sne	Schnee	sneeuw	snö
ice	is	Eis	ijs	is
smoke		Schmauch	smook	
reek	røg	Rauch	rook	rök
fire	fyr	Feuer	vuur	fyr
	ild			eld
ash	aske	Asche	as	aska
burn	brænde	brennen	branden	brinna
way	vej	Weg	weg	väg
road	red			red
berg	bjerg	Berg	berg	berg
bargh	bjerg	Berg	berg	berg
barrow	bjerg	Berg	berg	berg
berry	bjerg	Berg	berg	berg
red	rød	rot	rood	röd
green	grøn	grün	groen	grön
yellow		gelb	geel	
yellow		gehl	geel	
yellow		gel	geel	
yellow		gelb	geluw	
yellow		gehl	geluw	
yellow		gel	geluw	
	gul			gul
	gul			gål
white	hvid	weiß	wit	vit
swart	sort	schwarz	zwart	svart
swarth	sort	schwarz	zwart	svart
night	nat	Nacht	nacht	natt
day	dag	Tag	dag	dag
year	år	Jahr	jaar	år
warm	varm	warm	warm	varm
	lun			lugn
cold	kold	kalt	koud	kall
full	fuld	voll	vol	full
new	ny	neu	nieuw	ny
old		alt	oud	
eld		alt	oud	
	gammel		gammel	gammal
good	god	gut	goed	god
slight	slet	schlecht	slecht	slät
slight	slet	schlicht	slecht	slät
slight	slet	schlecht	slicht	slät

Continued

EN	DA	DE	NL	SV
slight	slet	schlicht	slicht	slät
	dårlig			dålig
foul	ful	faul	vuil	ful
stretch	strække	strecken	strekken	sträcka
right	ret	recht	recht	rätt
like	lig	gleich	gelijk	lik
round	rund	rund	rond	rund
sharp	skarp	scharf	scherp	skarp
	hvas			vass
dull	dval	toll	dol	
stump	stump	stumpf	stomp	stump
slow	sløv	schleh	slee	slö
slow	sløv	schleh	sleeuw	slö
smooth			smeuïg	
glad	glat	glatt	glad	glad
glad	glad	glatt	glad	glad
wet	våd			våt
		nass	nat	
		naß	nat	
	fugtig	feuchtig	vochtig	fuktig
	blød	bloß	bloot	blöt
dry	drøj	trocken	droog	dryg
	tør	dürr	dor	torr
correct	korrekt	korrekt	correct	korrekt
	rigtig	richtig	richtig	riktig
just	just	just	juist	just
near	nær		naar	när
nigh		nah	na	
nigh		nach	na	
tight	tæt	dicht	dicht	tät
by		bei	bij	bi
far	fjern	fern	ver	fjärran
	højre			höger
	venstre	winster		vänster
at	at			åt
at	ad			åt
on	å	an	aan	å
to		zu	toe	
to		zu	tot	
too		zu	toe	
too		zu	tot	
with	ved	wider	weder	vid
with	ved	wider	weer	vid
with	ved	wieder	weder	vid

Continued

EN	DA	DE	NL	SV
with	ved	wieder	weer	vid
in	i	in	in	i
mid	med	mit	met	med
mid	med	mit	mee	med
mid	med	mit	mede	med
and	end	und	en	än
and	end	und	ende	än
eke	og	auch	ook	och
if		ob	of	
umbe	om	um	om	om
umb	om	um	om	om
name	navn	Name	naam	namn
glass	glas	Glas	glas	glas
	slutte	schließen	sluiten	sluta
lock			luiken	lucka
lock	låg	Loch	lok	lock
louk	luge	liechen	lokken	lucka
louk	luge	locken	lokken	lucka
slot		Schüssel	sleutel	
	nøgle			nyckel
spoon	spån	Span	spaan	spån
		Löffel	lepel	
sheath	ske	Scheide	schede	sked
fork	fork	Forke	vork	
gavelock	gaffel	Gabel	gavel	gaffel
gavelock	gaffel	Gabel	gaffel	gaffel
knife	kniv			kniv
		Messer	mes	
flask	flaske	Flashe	fles	flaska
cheese		Käse	kaas	
	ost			ost
north	nord	Nord	noord	nord
east	øst	Ost	oost	
	øster	Osten	oosten	öster
south	syd	Süd	zuid	syd
west	vest	West	west	väst
		Westen	westen	väster
sheep		Schaf	schaap	
	får			får
horse		Ross	ros	russ
		Pferd	paard	
	hest	Hengst	hengst	häst
cow	ko	Kuh	koe	ko
goat	ged	Geiß	geit	get

Continued

EN	DA	DE	NL	SV
goose	gås	Gans	gans	gås
cat	kat	Katze	kat	katt
mouse	mus	Maus	muis	mus
rat	rotte	Ratte	rat	råtta
folk	folk	Volk	volk	folk
bed	bed	Bett	bed	bätt
	seng			säng
six	seks	sechs	zes	sex
seven	syv	sieben	zeven	sju
eight	otte	acht	acht	otta
nine	ni	neun	negen	nio
ten	ti	zehn	tien	tio
eleven	elleve	elf	elf	elva
twelve	tolv	zwölf	twaalf	tolv
king	konge	König	koning	konung
king	kong	König	koning	konung
king	konge	König	koning	kung
king	kong	König	koning	kung
thatch	tag	Dach	dak	tak
street		Straße	straat	stråt
gate	gade	Gasse	gas	gata
toy	tøj	Zeug	tuig	tyg

D. Expected probabilities

D.1 *P(A : B : C : D)*

D.1.1 As two-way transitions

$$
\begin{aligned}
P_e(A:B:C:D) &= \sqrt[n]{\prod_{i=1}^{n} \sqrt{\begin{aligned}&P_o(a_i:b_i)\cdot P_o(a_i:c_i)\cdot P_o(a_i:d_i)\\ &\quad \cdot P_o(b_i:c_i)\cdot P_o(b_i:d_i)\cdot P_o(c_i:d_i)\end{aligned}}} \\
&= \left(\prod_{i=1}^{n} \begin{aligned}&P_o(a_i:b_i)\cdot P_o(a_i:c_i)\cdot P_o(a_i:d_i)\\ &\quad \cdot P_o(b_i:c_i)\cdot P_o(b_i:d_i)\cdot P_o(c_i:d_i)\end{aligned}\right)^{\frac{1}{2n}} \\
&= \left(\begin{aligned}&\prod_{i=1}^{n} P_o(a_i:b_i)\cdot \prod_{i=1}^{n} P_o(a_i:c_i)\cdot \prod_{i=1}^{n} P_o(a_i:d_i)\\ &\quad \cdot \prod_{i=1}^{n} P_o(b_i:c_i)\cdot \prod_{i=1}^{n} P_o(b_i:d_i)\cdot \prod_{i=1}^{n} P_o(c_i:d_i)\end{aligned}\right)^{\frac{1}{2n}} \\
&= \sqrt{\left(\begin{aligned}&\prod_{i=1}^{n} P_o(a_i:b_i)\cdot \prod_{i=1}^{n} P_o(a_i:c_i)\cdot \prod_{i=1}^{n} P_o(a_i:d_i)\\ &\quad \cdot \prod_{i=1}^{n} P_o(b_i:c_i)\cdot \prod_{i=1}^{n} P_o(b_i:d_i)\cdot \prod_{i=1}^{n} P_o(c_i:d_i)\end{aligned}\right)^{\frac{1}{n}}} \\
&= \sqrt{\begin{aligned}&\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i)}\cdot \sqrt[n]{\prod_{i=1}^{n} P_o(a_i:c_i)}\cdot \sqrt[n]{\prod_{i=1}^{n} P_o(a_i:d_i)}\\ &\quad \cdot \sqrt[n]{\prod_{i=1}^{n} P_o(b_i:c_i)}\cdot \sqrt[n]{\prod_{i=1}^{n} P_o(b_i:d_i)}\cdot \sqrt[n]{\prod_{i=1}^{n} P_o(c_i:d_i)}\end{aligned}} \\
&= \sqrt{\begin{aligned}&P_o(A:B)\cdot P_o(A:C)\cdot P_o(A:D)\\ &\quad \cdot P_o(B:C)\cdot P_o(B:D)\cdot P_o(C:D)\end{aligned}}
\end{aligned}
\tag{D.1}
$$

D.1.2 As three-way transitions

$$
\begin{aligned}
P_e(A:B:C:D) &= \sqrt[n]{\prod_{i=1}^{n} \sqrt[8]{P_o(a_i:b_i:c_i)^3 \cdot P_o(a_i:b_i:d_i)^3 \cdot P_o(a_i:c_i:d_i)^3 \cdot P_o(b_i:c_i:d_i)^3}} \\
&= \left(\prod_{i=1}^{n} P_o(a_i:b_i:c_i) \cdot P_o(a_i:b_i:d_i) \cdot P_o(a_i:c_i:d_i) \cdot P_o(b_i:c_i:d_i) \right)^{\frac{3}{8n}} \\
&= \left(\begin{array}{l} \prod_{i=1}^{n} P_o(a_i:b_i:c_i) \cdot \prod_{i=1}^{n} P_o(a_i:b_i:d_i) \\ \quad \cdot \prod_{i=1}^{n} P_o(a_i:c_i:d_i) \cdot \prod_{i=1}^{n} P_o(b_i:c_i:d_i) \end{array} \right)^{\frac{3}{8n}} \\
&= \sqrt[8]{\left(\begin{array}{l} \prod_{i=1}^{n} P_o(a_i:b_i:c_i) \cdot \prod_{i=1}^{n} P_o(a_i:b_i:d_i) \\ \quad \cdot \prod_{i=1}^{n} P_o(a_i:c_i:d_i) \cdot \prod_{i=1}^{n} P_o(b_i:c_i:d_i) \end{array} \right)^{\frac{3}{n}}} \\
&= \sqrt[8]{\begin{array}{l} \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i:c_i)} \right)^3 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i:d_i)} \right)^3 \\ \quad \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:c_i:d_i)} \right)^3 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(b_i:c_i:d_i)} \right)^3 \end{array}} \\
&= \sqrt[8]{P_o(A:B:C)^3 \cdot P_o(A:B:D)^3 \cdot P_o(A:C:D)^3 \cdot P_o(B:C:D)^3}
\end{aligned}
\tag{D.2}
$$

D.2 *P(A : B : C : D : E)*

D.2.1 As two-way transitions

$$
\begin{aligned}
P_e(A:B:C:D:E) &= \sqrt[n]{\prod_{i=1}^{n} \sqrt[5]{\begin{array}{c} P_o(a_i:b_i)^2 \cdot P_o(a_i:c_i)^2 \cdot P_o(a_i:d_i)^2 \cdot P_o(a_i:e_i)^2 \\ \cdot P_o(b_i:c_i)^2 \cdot P_o(b_i:d_i)^2 \cdot P_o(b_i:e_i)^2 \\ \cdot P_o(c_i:d_i)^2 \cdot P_o(c_i:e_i)^2 \cdot P_o(d_i:e_i)^2 \end{array}}} \\
&= \left(\prod_{i=1}^{n} \begin{array}{c} P_o(a_i:b_i) \cdot P_o(a_i:c_i) \cdot P_o(a_i:d_i) \cdot P_o(a_i:e_i) \\ \cdot P_o(b_i:c_i) \cdot P_o(b_i:d_i) \cdot P_o(b_i:e_i) \\ \cdot P_o(c_i:d_i) \cdot P_o(c_i:e_i) \cdot P_o(d_i:e_i) \end{array} \right)^{\frac{2}{5n}} \\
&= \left(\begin{array}{c} \prod_{i=1}^{n} P_o(a_i:b_i) \cdot \prod_{i=1}^{n} P_o(a_i:c_i) \cdot \prod_{i=1}^{n} P_o(a_i:d_i) \cdot \prod_{i=1}^{n} P_o(a_i:e_i) \\ \cdot \prod_{i=1}^{n} P_o(b_i:c_i) \cdot \prod_{i=1}^{n} P_o(b_i:d_i) \cdot \prod_{i=1}^{n} P_o(b_i:e_i) \\ \cdot \prod_{i=1}^{n} P_o(c_i:d_i) \cdot \prod_{i=1}^{n} P_o(c_i:e_i) \cdot \prod_{i=1}^{n} P_o(d_i:e_i) \end{array} \right)^{\frac{2}{5n}} \\
&= \sqrt[5]{\left(\begin{array}{c} \prod_{i=1}^{n} P_o(a_i:b_i) \cdot \prod_{i=1}^{n} P_o(a_i:c_i) \cdot \prod_{i=1}^{n} P_o(a_i:d_i) \cdot \prod_{i=1}^{n} P_o(a_i:e_i) \\ \cdot \prod_{i=1}^{n} P_o(b_i:c_i) \cdot \prod_{i=1}^{n} P_o(b_i:d_i) \cdot \prod_{i=1}^{n} P_o(b_i:e_i) \\ \cdot \prod_{i=1}^{n} P_o(c_i:d_i) \cdot \prod_{i=1}^{n} P_o(c_i:e_i) \cdot \prod_{i=1}^{n} P_o(d_i:e_i) \end{array} \right)^{\frac{2}{n}}} \\
&= \sqrt[5]{\begin{array}{c} \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i)}\right)^2 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:c_i)}\right)^2 \\ \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:d_i)}\right)^2 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:e_i)}\right)^2 \\ \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(b_i:c_i)}\right)^2 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(b_i:d_i)}\right)^2 \\ \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(b_i:e_i)}\right)^2 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(c_i:d_i)}\right)^2 \\ \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(c_i:e_i)}\right)^2 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(d_i:e_i)}\right)^2 \end{array}}
\end{aligned}
\tag{D.3}
$$

$$P_e(A:B:C:D:E) = \sqrt[5]{\begin{aligned} & P_o(A:B)^2 \cdot P_o(A:C)^2 \cdot P_o(A:D)^2 \cdot P_o(A:E)^2 \cdot P_o(B:C)^2 \\ & \quad \cdot P_o(B:D)^2 \cdot P_o(B:E)^2 \cdot P_o(C:D)^2 \cdot P_o(C:E)^2 \cdot P_o(D:E)^2 \end{aligned}} \tag{D.3}$$

D.2.2 As three-way transitions

$$\begin{aligned} P_e(A:B:C:D:E) &= \sqrt[n]{\prod_{i=1}^{n} \sqrt[5]{\begin{aligned} & P_o(a_i:b_i:c_i) \cdot P_o(a_i:b_i:d_i) \cdot P_o(a_i:b_i:e_i) \cdot P_o(a_i:c_i:d_i) \\ & \quad \cdot P_o(a_i:c_i:e_i) \cdot P_o(a_i:d_i:e_i) \cdot P_o(b_i:c_i:d_i) \\ & \quad \cdot P_o(b_i:c_i:e_i) \cdot P_o(b_i:d_i:e_i) \cdot P_o(c_i:d_i:e_i) \end{aligned}}} \\ &= \left(\prod_{i=1}^{n} \begin{aligned} & P_o(a_i:b_i:c_i) \cdot P_o(a_i:b_i:d_i) \cdot P_o(a_i:b_i:e_i) \cdot P_o(a_i:c_i:d_i) \\ & \quad \cdot P_o(a_i:c_i:e_i) \cdot P_o(a_i:d_i:e_i) \cdot P_o(b_i:c_i:d_i) \\ & \quad \cdot P_o(b_i:c_i:e_i) \cdot P_o(b_i:d_i:e_i) \cdot P_o(c_i:d_i:e_i) \end{aligned} \right)^{\frac{1}{5n}} \\ &= \left(\begin{aligned} & \prod_{i=1}^{n} P_o(a_i:b_i:c_i) \cdot \prod_{i=1}^{n} P_o(a_i:b_i:d_i) \cdot \prod_{i=1}^{n} P_o(a_i:b_i:e_i) \\ & \quad \cdot \prod_{i=1}^{n} P_o(a_i:c_i:d_i) \cdot \prod_{i=1}^{n} P_o(a_i:c_i:e_i) \cdot \prod_{i=1}^{n} P_o(a_i:d_i:e_i) \\ & \quad \cdot \prod_{i=1}^{n} P_o(b_i:c_i:d_i) \cdot \prod_{i=1}^{n} P_o(b_i:c_i:e_i) \\ & \quad \cdot \prod_{i=1}^{n} P_o(b_i:d_i:e_i) \cdot \prod_{i=1}^{n} P_o(c_i:d_i:e_i) \end{aligned} \right)^{\frac{1}{5n}} \\ &= \sqrt[5]{\left(\begin{aligned} & \prod_{i=1}^{n} P_o(a_i:b_i:c_i) \cdot \prod_{i=1}^{n} P_o(a_i:b_i:d_i) \cdot \prod_{i=1}^{n} P_o(a_i:b_i:e_i) \\ & \quad \cdot \prod_{i=1}^{n} P_o(a_i:c_i:d_i) \cdot \prod_{i=1}^{n} P_o(a_i:c_i:e_i) \cdot \prod_{i=1}^{n} P_o(a_i:d_i:e_i) \\ & \quad \cdot \prod_{i=1}^{n} P_o(b_i:c_i:d_i) \cdot \prod_{i=1}^{n} P_o(b_i:c_i:e_i) \\ & \quad \cdot \prod_{i=1}^{n} P_o(b_i:d_i:e_i) \cdot \prod_{i=1}^{n} P_o(c_i:d_i:e_i) \end{aligned} \right)^{\frac{1}{n}}} \end{aligned} \tag{D.4}$$

$$
\begin{aligned}
P_e(A:B:C:D:E) &= \sqrt[5]{\begin{aligned}&\left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i:c_i)}\right)\cdot\left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i:d_i)}\right)\\ &\quad\cdot\left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i:e_i)}\right)\cdot\left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:c_i:d_i)}\right)\\ &\quad\cdot\left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:c_i:e_i)}\right)\cdot\left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:d_i:e_i)}\right)\\ &\quad\cdot\left(\sqrt[n]{\prod_{i=1}^{n} P_o(b_i:c_i:d_i)}\right)\cdot\left(\sqrt[n]{\prod_{i=1}^{n} P_o(b_i:c_i:e_i)}\right)\\ &\quad\cdot\left(\sqrt[n]{\prod_{i=1}^{n} P_o(b_i:d_i:e_i)}\right)\cdot\left(\sqrt[n]{\prod_{i=1}^{n} P_o(c_i:d_i:e_i)}\right)\end{aligned}} \\
&= \sqrt[5]{\begin{aligned}&P_o(A:B:C)\cdot P_o(A:B:D)\cdot P_o(A:B:E)\cdot P_o(A:C:D)\\ &\quad\cdot P_o(A:C:E)\cdot P_o(A:D:E)\cdot P_o(B:C:D)\\ &\quad\cdot P_o(B:C:E)\cdot P_o(B:D:E)\cdot P_o(C:D:E)\end{aligned}}
\end{aligned}
\tag{D.4}
$$

D.2.3 As four-way transitions

$$
\begin{aligned}
P_e(A:B:C:D:E) &= \sqrt[n]{\prod_{i=1}^{n}\sqrt[15]{\begin{gathered}P_o(a_i:b_i:c_i:d_i)^4\cdot P_o(a_i:b_i:c_i:e_i)^4\cdot P_o(a_i:b_i:d_i:e_i)^4\\ \cdot P_o(a_i:c_i:d_i:e_i)^4\cdot P_o(b_i:c_i:d_i:e_i)^4\end{gathered}}} \\
&= \left(\prod_{i=1}^{n}\begin{gathered}P_o(a_i:b_i:c_i:d_i)\cdot P_o(a_i:b_i:c_i:e_i)\cdot P_o(a_i:b_i:d_i:e_i)\\ \cdot P_o(a_i:c_i:d_i:e_i)\cdot P_o(b_i:c_i:d_i:e_i)\end{gathered}\right)^{\frac{4}{15n}} \\
&= \left(\begin{aligned}&\prod_{i=1}^{n} P_o(a_i:b_i:c_i:d_i)\cdot\prod_{i=1}^{n} P_o(a_i:b_i:c_i:e_i)\\ &\quad\cdot\prod_{i=1}^{n} P_o(a_i:b_i:d_i:e_i)\cdot\prod_{i=1}^{n} P_o(a_i:c_i:d_i:e_i)\\ &\qquad\cdot\prod_{i=1}^{n} P_o(b_i:c_i:d_i:e_i)\end{aligned}\right)^{\frac{4}{15n}}
\end{aligned}
\tag{D.5}
$$

$$
\begin{aligned}
P_e(A:B:C:D:E) &= \sqrt[15]{\left(\begin{array}{l}\prod_{i=1}^{n} P_o(a_i:b_i:c_i:d_i) \cdot \prod_{i=1}^{n} P_o(a_i:b_i:c_i:e_i) \\ \quad \cdot \prod_{i=1}^{n} P_o(a_i:b_i:d_i:e_i) \cdot \prod_{i=1}^{n} P_o(a_i:c_i:d_i:e_i) \\ \qquad \cdot \prod_{i=1}^{n} P_o(b_i:c_i:d_i:e_i)\end{array}\right)^{\frac{4}{n}}} \\
&= \sqrt[15]{\begin{array}{l}\left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i:c_i:d_i)}\right)^4 \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i:c_i:e_i)}\right)^4 \\ \quad \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:b_i:d_i:e_i)}\right)^4 \cdot \left((\sqrt[n]{\prod_{i=1}^{n} P_o(a_i:c_i:d_i:e_i)}\right)^4 \\ \qquad \cdot \left(\sqrt[n]{\prod_{i=1}^{n} P_o(b_i:c_i:d_i:e_i)}\right)^4\end{array}} \\
&= \sqrt[15]{\begin{array}{c}P_o(A:B:C:D)^4 \cdot P_o(A:B:C:E)^4 \cdot P_o(A:B:D:E)^4 \\ \cdot P_o(A:C:D:E)^4 \cdot P_o(B:C:D:E)^4\end{array}}
\end{aligned} \tag{D.5}
$$