Date of acceptance Grade

Instructor

Minimum Description Length Models for Unsupervised Learning of Morphology

Javad Nouri

Helsinki May 31, 2016

MSc Thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

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

Tekijä — Författare — Author
Javad Nouri

Työn nimi — Arbetets titel — Title
Minimum Description Length Models forUnsupervised Learning of Morphology

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
MSc Thesis	May 31, 2016	69 pages + 6 appendices

Tiivistelmä — Referat — Abstract

This thesis work introduces an approach to unsupervised learning of morphological structure of human languages. We focus on morphologically rich languages and the goal is to construct a knowledge-free and language-independent model. This model works by receiving a long list of words in a language and is expected to learn how to segment the input words in a way that the resulting segments correspond to morphemes in the target language. Several improvements inspired by well-motivated linguistic principles of morphology of languages are introduced to the proposed MDL-based learning algorithm.

In addition to the learning algorithm, a new evaluation method and corresponding resources are introduced. Evaluation of morphological segmentations is a challenging task due to the inherent ambiguity of natural languages and underlying morphological processes such as fusion which encumber identification of unique "correct" segmentations for words. Our evaluation method addresses the problem of segmentation evaluation with a focus on consistency of segmentations.

Our approach is tested on data from Finnish, Turkish, and Russian. Evaluation shows a gain in performance over the state of the art.

ACM Computing Classification System (CCS):

E. Data
  E.4 Coding and Information Theory
G. Mathematics of Computing
  G.3 Probability and Statistics
I. Computing Methodologies
  I.2 Artificial Intelligence
    I.2.6 Learning
    I.2.7 Natural Language Processing

Avainsanat — Nyckelord — Keywords
minimum description length principle, morphology, morphological segmentation

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

Morphology (from ancient Greek "μορφή" (*form*) + -logy) is defined as the scientific study of *form*. In linguistics, morphology tries to identify, analyze and describe processes for generating words in a natural language, or it can be defined as the study of *forms of words*, where one studies the construction of words; a subfield of linguistics that is said to have been given less credit than it deserves [Mat74].

This thesis focuses on methods for learning morphology of a given natural language in an *unsupervised* fashion. The term morphology is broad in this sense and one should make it clear what is meant by learning the *morphology* of a language.

From an analytic point of view, morphology could be defined as analyzing a given word of the language by describing the underlying processes the word at hand is formed by. From a synthetic point of view, it can be defined as generating the proper word form based on the *stem*[1] and the sense in which the word is going to be used and the role it will take in the sentence. From a descriptive point of view, morphology can be defined as scientific expression of the word formation processes of the language, possibly in form of a set of rules or a grammar. These three points of view are, of course, closely related.

In this thesis, we focus on finding the *correct segmentation* of words into their *morphs*—word segments that represent the smallest meaningful units of the language, based on only a list of words of the language and without any further information about the language at hand.[2] This is what the *unsupervised* part reflects. We want to verify whether it is possible to learn the underlying *word formation rules* of the language solely based on the words and possibly well understood universal linguistic principles. In other words, whether the morphological system is *inherently encoded* in the language.

Although answering the question of whether the structure of words of a language can be explained only based on a list of words from that language, which helps answer questions about human languages [AAAK04], is a fascinating problem in itself, there are other practical motivations for developing such tools as well. Some of these motivations are briefly explored here.

There have been many efforts to create automatic morphological analyzers for hu-

[1] Stem is defined as a form of the word that cannot be further analyzed.

[2] In the literature this is sometimes reflected as *knowledge-free*, however in this thesis independence from language or being knowledge-free is implied when discussing unsupervised methods.

man languages [PO09, Çöl10]. Such tools, given a word in a language, will output the base form of the word along with several morphological tags. Morphological analyzers are used in many natural language processing tasks such as parsing, machine translation, speech recognition, information retrieval, etc. Construction of automatic morphological analyzers is often a labor-intensive task which requires considerable amount of expert knowledge. As a result of this, such projects are often very expensive in terms of human labor and no such analyzers have been created for many of the underresourced languages. A substantial amount of subjectivity is involved in the manual creation of analyzers and two different analyzers for the same language might result in different way of modeling the morphology of the language,[3] hence different analyses for the same input word.

Supervised methods for morphology learning can ease this process, but they also need expert knowledge in form of annotated data to achieve the goal. Unsupervised and knowledge-free (language-independent) methods for inferring morphological processes of languages, if proved successful, can play a crucial role in creation of analyzers, since they eliminate the need for huge expert knowledge.

Morphological analyzers are not the only tools that benefit from these unsupervised methods. For example compilation of MRDs (Machine-Readable Dictionaries) which are intended to include every possible words that one is ever likely to encounter is an example of other resources that can take advantage of unsupervised learning methods [SJ00]. It has been stated that "the quest for an efficient method for the analysis and generation of word-forms is no longer an academic research topic" [KK97].

In the rest of this chapter, after a brief introduction to unsupervised machine learning, we give an introductory review of the *minimum description length principle* (MDL), which is the core of our approach. The thesis is structured as follows: in Chapter 2 the linguistic terminology used throughout the thesis is briefly reviewed followed by definition of the problem; Chapter 3 reviews some of the prior research related to this work; in Chapter 4 our method to address the problem is explained in detail; in Chapter 5 we present a new approach to evaluation of the performance of the method, and corresponding gold-standard annotation guidelines;[4] Chapter 6 reviews several linguistically inspired modifications and extensions to the algorithm followed by results of the system tested on Finnish, Turkish, and Russian, as well as comparison of the results to a state-of-the-art method: Morfessor CatMAP; the

[3]Even if they use the same technology.

[4]Parts of this work have been published elsewhere.

thesis is concluded with a discussion of the important features of this work and ideas about potential future work in Chapter 7.

1.1 Unsupervised Machine Learning

Machine learning is a subfield of computer science that deals with learning from data. Attempts to use computers for learning are not new and go back to as early as 1960s. With the ever growing amount of data, machine learning methods are constantly improving and the role of learning from data is growing more and more. Machine learning methods try to build a model based on observations (data) which capture and discover patterns in data and possibly perform predictions for future instances.

Machine learning methods are primarily categorized into *supervised* and *unsupervised* methods, based on the task they are expected to perform and the type of data they deal with. If the data is labeled, meaning that there are output values for each data item that will not be available for future data, and the task is to discover a mapping from input data to the output values, the task is said to be supervised. The most well-known example of supervised machine learning problems is the problem of *classification*, in which data items are divided into multiple classes and the task is to find underlying rules that help predict the class for future unlabeled input data.

Unsupervised machine leaning deals with raw data that are not labeled in any way and the goal is to discover patterns in the data. A very well-known unsupervised learning problem is *clustering*, in which the goal is to divide the input data into several groups which are not known beforehand. The problem we address in this thesis work is of unsupervised nature; we deal with a list of words of a language, without any further information and the goal is to find underlying morphological processes and rules and learn to segment them in a linguistically-sensible fashion, without providing the algorithm with any segmentations.

1.2 The Minimum Description Length Principle

The minimum description length principle, abbreviated as MDL, which was introduced by Jorma Rissanen in 1978 [Ris78], is a formalization of *Occam's razor* and provides a generic solution to the problem of model selection [Grü07]. Suppose that we have several models that try to explain a set of limited observations and one

needs to choose one of the models which better explains the data. This is the problem of model selection. Model selection is one of the most important questions in statistical inference and data modeling.

According to the MDL principle, the best model is the one that compresses the data the most. Viewing the problem of learning as a data compression task is the core of the MDL principle. The ultimate goal of learning is to find the underlying *rules* and processes that help describe the data; the more rules one can discover from the data, the better the data will be described. The amount of *regularities* one can find in the data is also in correlation with how much that data can be compressed; finding more rules results in shorter description of the data.

The length of the shortest computer program that can be written to print out the data is called the *Kolmogorov complexity* of the data [Kol63, Grü07]. In general, the more complex the data is, the higher its Kolmogorov complexity will be. Kolmogorov complexity is closely related to data entropy defined by Shannon [Sha01]. Kolmogorov complexity is defined relative to a programming language in which the program is written. This means that it depends on the chosen programming language. It has been proven that Kolmogorov complexity is not computable. This has been done by showing that no program P can be constructed that given data D will return the shortest program that produces D [Grü07]. Although the *invariance theorem* states that Kolmogorov complexity of data D relative two programming languages L_1 and L_2 differ only in a constant number of bits provided that data D is long enough, because of aforementioned problems Kolmogorov complexity is not useful in practice [Grü07].

The MDL principle provides a practical solution to model selection in absence of practicality of Kolmogorov complexity. Given a model class from which a model should be selected, choose the one that yields the best compression of the data. MDL has several features that makes it distinct from other approaches. MDL automatically avoids overfitting by trading off the goodness-of-fit and the model complexity (by formalizing Occam's razor) and does not require other *ad hoc* modifications to avoid overfitting. Also in general the MDL-based methods, unlike many statistical methods, do not assume any specific *true* processes that generate the data. In many cases such *"underlying truth"* does not even exist [Grü07].

MDL is closely related to Bayesian methods and also is believed to select models with good *prediction* abilities of *unseen* data. MDL is closely related to the concept of *universal codes*. A universal code for a model is the code that represents the

data almost as compactly as the best model, i.e. the model that compresses the data the most. One type of such codes is the two-part MDL code in which the code length for the data consists of two parts: $L(M)$—code-length of the model— and $L(D|M)$—code-length of data given the model— This allows automatic tradeoff between model complexity (by which $L(M)$ grows) and goodness of fit (by which $L(D|M)$ decreases), hence avoiding overfitting. Other types of universal codes are Bayesian universal codes, normalized maximum likelihood (NML) universal codes, prequential plug-in universal codes which are used in this work and will be discussed further in Section 4.1, etc [Grü07].

2 Problem Definition

In this chapter, after a brief linguistic introduction and review of some linguistic terms used in the thesis, we define the problem that is being addressed in this work.

2.1 Terminology

In this section we define the (mostly linguistic) terminology that will be used throughout the thesis. The linguistic terminology and detailed discussions of morphology can be found in [Mat74] and [AF11].

- Word: A **word** is an independent and meaningful unit of speech or writing, which usually is accompanied by other words to form sentences. Some examples in English are *dog*, *football*, *dishes*, etc.
- Morph: A **morph** is a segment of the word that represents the smallest meaningful unit of the language and cannot be further split into other meaningful units. For example the English word *dogs* consists of two morphs: *dog* (an animal) + *s* (signifying more than one dog).
- Morpheme: A **morpheme** is the smallest meaningful unit of the language. Morpheme is an abstract notion and can be defined as a set of morphs that correspond to the same meaningful unit. Morphs are realizations of morphemes in the words. For example the first morphs of the Finnish words *kansi*, *kannen*, *kanteen*, and *kantta* are all realizations of one morpheme: {kansi, kanne-, kante-, kant-}—meaning: cover, lid, cap.

It is also crucial to distinguish morphemes from words, as morphemes do not always appear as words. In fact a morpheme may or may not appear on its own in the language, but a word is by definition an independent unit. Morphemes can be classified to be *free* or *bound*. Free morphemes can appear in the language as an independent word, whereas bound morphemes have to be accompanied by at least one free morpheme to form a word. For example the English word *unattractive* consists of three morphemes: *un-* is a bound morpheme which signifies negation of the next adjective or past participle, *attract* is a free morpheme,[5] and *-ive* is a bound morpheme for forming adjectives from verbs.

- Allomorphy: **Allomorphs** are morphemes that are semantically the same, but differ in pronunciation and perhaps in writing as well. A simple example of allomorphy in English is the morpheme for forming plural of nouns which is a */z/* in *dogs*, but manifests itself as */ɪz/* in *ashes*. Depending on the target language there are different types of allomorphs. For example vowel harmony in Finnish prescribes at least two allomorphs for most suffixes, such as *"-ton"* and *"-tön"* for indication of lack of something respectively for words containing back and front vowels. In Turkic languages the same principle of vowel harmony results in four variants of some suffixes.[6]

- Inflection: **Inflection** is a morphological process which involves addition of an *inflectional* morpheme to a base. Inflectional morphemes create new words from the base, but do not change the semantics or part of speech of the base word. For example it can change the tense, person, or number of a verb, or it can be used to form the plural of nouns. An example in English is the morpheme for forming plurals (written as "s") which is used to form *dogs* from *dog*.

[5]More accurately, if one wants to account the history of the words in ancestor languages (or source languages in case of borrowings) one could break *attract* into two morphemes, since *attract* descends form Latin *attractus* which is the past participle of *attrahere* (to draw to), which is in turn formed by Latin prefix *ad-* (to) + *trahere* (to draw). This exhibits one potential source of subjectivity in describing the morphology of languages. One could argue that the word is old enough in English to forget the historical structure of it since it would not help much in practical applications of the morphological analysis, whereas others might want to include it because it reveals more information about the language and the word than *attract* by itself does.

[6]For instance the analogous suffix for indicating lack of something in Turkish has 4 variants: *-sız*, *-siz*, *-suz*, or *-süz* depending on the position of tongue (front or back) and roundedness of the vowels in the preceding morpheme.

- Derivation: As opposed to inflectional processes, **derivation** is a morphological process in which the semantic or part of speech of the base words may change. It can be said that these processes create new words, whereas inflection only changes the form of the word. An example that we encountered earlier is adding the suffix *"-ive"* in English which is used to create adjectives out of other words, such as *attractive, collective, massive, constructive*, etc. Although the resulting words are semantically related to the base words, they do not exactly signify the same thing and usually the meaning of the word changes.

- Compounding: **Compounds** are words that have more than a free morpheme or root in them. Some languages such as German and Finnish make extensive use of compounding. For example the word *football* consists of two free morphemes *foot* + *ball*. Other examples that use more complex morphological structures are *Götterspeise* in German and *talvikunnossapitoa* in Finnish.

2.2 Morphological Typology

Morphological typology refers to a way of classification of natural languages based on their morphological structures.

Generally, languages are divided into two major groups: **analytic languages** and **synthetic languages**. Analytic languages are the ones that use little inflection in their structure and instead use other non-morphological mechanisms such as word order to clarify the underlying role of the words. On the other hand, **synthetic** languages are the ones that have a higher morpheme per word ratio. In other words, one word can appear in many different inflected forms to reflect the role it is taking in the sentence. Synthetic languages usually trade this added complexity off by removing word order constraints, thus the order of words in a sentence can be chosen more freely.

There have been efforts to classify synthetic languages based on how morphemes are combined together to form a word. Synthetic languages are categorized as **agglutinative** or **fusional**. Agglutinative languages, also known as *concatenative* languages, use *agglutination* as the primary approach to morpheme combination. In this type of languages, the morphemes remain mostly unchanged when combined to form a new word, as if they have been concatenated and the word is a sequence of morphemes. This is in contrast to *fusional* languages where there is fusion in com-

bining the morphemes and it may be difficult to distinguish morpheme boundaries. Although scholars tend to categorize languages in this way, it is essential to note that it is not plausible to refer to a language as *fully agglutinative* or *fully fusional* since languages employ both types of processes in their structure to some extent and care must be taken when categorizing languages as such.

In contrast to concatenative languages, some languages are known to be non-concatenative. These languages are said to use discontinuous morphology in which words are formed by other means that stringing morphemes together [HS13]. One example is Arabic which uses *transfixation* for morphological generation. In Arabic, certain vowels and consonants are added in between the consonants of the root to create new morphemes. One example of such a process can be found in Figure 1. Other non-concatenative morphological processes include *ablaut*, *reduplication*, and *truncation*.

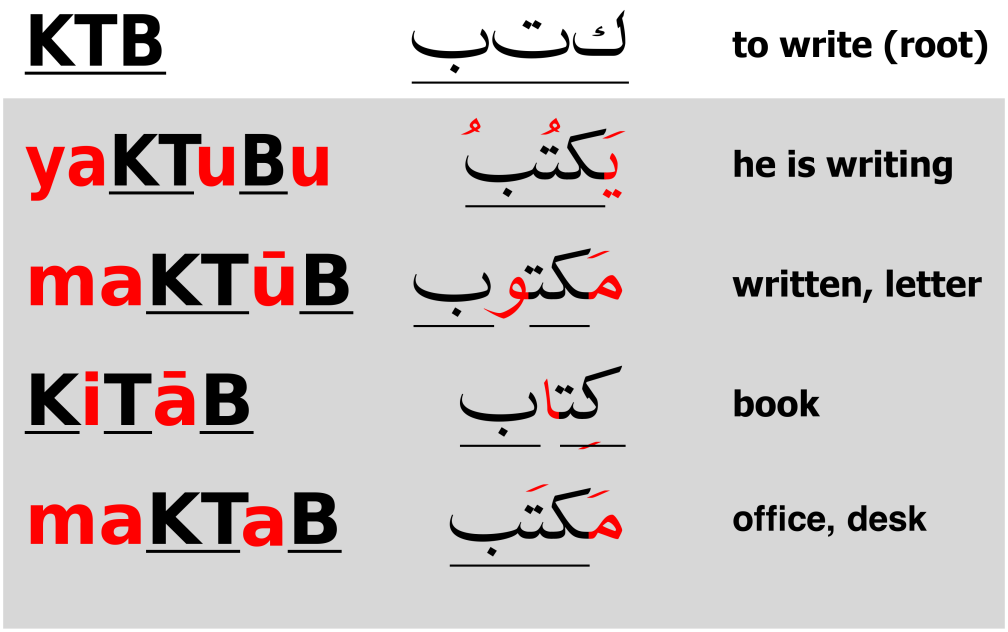

Figure 1: Example of Arabic transfixation from a three-consonant root. The root is shown in the first row and root symbols are underlined in each word. New words are created by placing the root in certain patterns, which can put sounds anywhere between or around the root consonants. The vowels and consonants of the pattern are shown in red (not underlined, short vowels appear as diacritics in red above or under the consonants). As an example, the template *"ya12u3u"* will result in the masculine 3rd person present tense verb, where 1, 2, and 3 denote the first, second and the third consonant of the root, respectively.

In this work we focus on concatenative morphological systems. A more universal system that can learn the morphology of non-concatenative languages is in our future work plans.

2.3 Input Data

The input to our algorithm is a list of distinct words of the target language. We do not use any information about the context of the words, nor any information about the frequency of their usage; adapting the model to use such information is part of the future plans.

Although morphology of the language might be better reflected in the spoken form of the language, we use the written form of the words, since it is the easiest form to collect data for, and gives an approximation of the spoken form. We collect and preprocess the data from books that are freely available in digital format for public use. The list of words are then extracted from these books.

2.4 Morphological Segmentation

For languages that are considered to be *agglutinative* or mostly so, one could define a *segmentation* of a word into morphs. If the language at hand were fully agglutinative without more complex morphophonological phenomena such as fusion, segmenting the words by a human expert would be trivial, since it would be clear which morph each letter corresponds to. For instance the Finnish word *talossa*—meaning "in the house", can be unambiguously segmented as *talo* (house) + *-ssa* (inessive suffix). But none of the natural languages are fully agglutinative and each one shows at least several more complex phenomena which leaves the definition of segmentation somehow subjective. For instance in case of the Finnish word *yrityksessä*—meaning "in the company", due to fusion, it is not clear what the segments should be; *yrityks+essä* or *yritykse+ssä*, or perhaps some other segmentation.

We define *segmentation* of a word as slicing the word in a way that each piece is a morphologically valid morph as defined in Section 2.1. To each of these segments can be assigned a *label* which reflects which morpheme it corresponds to. One might accurately argue that in languages that are not fully agglutinative, for a given word, there might be more than one *reasonable* segmentation. This is in general true since for that kind of processes the symbols closer to the morpheme boundaries could be

considered as part of either morph.

In this thesis we are looking for a method for reasonable segmentation of the set of words, giving the model freedom to make decisions under such circumstances, provided that the model remains consistent in the decisions it makes. If a system were evaluated based on one of these alternatives as the *correct* segmentation, it would not be fair to the model since it might have obtained a set of alternative, equally good segmentations. This problem of evaluation and how we address it will be explained in details in Chapter 5.

We want a method, that given a set of words in a language (and no further a priori information about the language, except for possibly universal rules which apply to all languages), will learn to segment each word into strings that in the ideal case meet the following conditions:

1. Each string is a morph that corresponds to a linguistically meaningful morpheme.
2. Concatenating the segments will result in the original word.

These are the two requirements that this work focuses on. Furthermore, the model can choose to:

- Assign a label to each morph which gives more information about the morpheme it corresponds to
- Determine other aspects of morphological processes of the underlying language

The latter two points, if addressed, should draw the model closer to full morphological analysis of the language, however, full morphological analysis is not our goal in this work.

Some examples of segmentations for English and Finnish are listed in Figure 2.

3 Related Work

There has been substantial amount of research in unsupervised morphological learning during the past decades.

Morpho Challenge is a series of events organized for designing statistical machine learning methods for learning morphology [KCV+06, KCV07, KTV08, KVT+09,

Word	Segmentation
joukoista	jouko+i+sta
kokoaisin	koko+a+isi+n
mielitelkö	miel+it+el+kö
puolustautua	puol+us+t+a+utu+a
sanoitte	sano+i+tte
taajain	taaja+i+n
talossa	talo+ssa
virroissa	virr+o+i+ssa

(a) Finnish

Word	Segmentation
attractive	attract+ive
cats	cat+s
dogs	dog+s
formalize	form+al+ize
recalculate	re+calculate
strengthened	strength+en+ed
unspeakable	un+speak+able
walking	walk+ing

(b) English

Figure 2: Examples of word segmentation for Finnish and English.

KVTL10]. Data-sets for Finnish, English, Turkish, German, and Arabic (vowelized and non-vowelized)[7] are provided by the organizers for training and evaluating unsupervised and semi-supervised morphology learning methods.

In-depth surveys of morphological learning methods can be found in [HB11] and [CM14]. In this chapter we review some of the works that have addressed morphology learning and general methods that have been employed to address this problem. [HB11] categorizes the efforts into four fundamentally different approaches:

Border and frequency methods are the ones that utilize frequency of adjacent substrings as evidence of morpheme boundaries. In other words, if a substring is adjacent to many different substrings, it is more likely to be a morpheme. High frequency of substrings themselves also suggests them as morpheme candidates.

Group and Abstract methods approach the problem by first grouping (clustering) the words into morphologically related sets. This is often done using some heuristic such as string edit distance, semantics, etc. These groups are then looked up by an algorithm for recurrent patterns of morphological processes.

Features and Classes methods try to represent words as sets of features instead of their written representation. This way features that occur rarely represent a stem and more frequent features are more likely to represent affixes. This in general is true since affixes tend to occur much more frequently than stems.

[7] Arabic uses a consonant-based writing system in which most vowels are omitted in writing. Vowelization is a way of rewriting the words along with the vowels that are omitted in normal writing.

Phonological Categories and Separation methods categorize phonemes of the words into classes such as vowels, consonants, etc. These techniques are then used to learn the patterns present in the words. To follow this approach, one needs the phonemes of the words which are generally approximated by graphemes—the smallest units in the writing system that in many writing systems are used to represent sounds. It is worth mentioning that this approach is designed to work with *intercalated* morphological systems such as Arabic which follow non-concatenative processes.

[CM14] on the other hand classifies the methods for unsupervised morphological learning into a finer list based on what machine learning approach they follow.

LSV (Letter Successor Variety) models: Harris's [Har55] LSV-based word segmentation algorithm is among the first attempts for unsupervised word segmentation. These methods work by constructing a *trie* [De 59] of the input words. The number of letters that can follow a letter is called the *successor variety* of that letter. Symmetrically the number of letters preceding a letter is called the *predecessor varieties* of the letter. These numbers are then used to determine word segment boundaries or so called split points. This approach is similar to the *border and frequency* methods in the earlier typology of algorithms. An illustration of how this family of algorithms work can be found in Figure 3.

MDL models rely on Rissanen's *minimum description length* principle [Ris78]. MDL principle states that the shorter the code length a model assigns to the data, the better that model describes the data. Our method is also in this family.

ML (Maximum likelihood): In Bayesian statistics, likelihood of model M given the data D is defined as the probability of data D given model M. Sometimes for ease of calculation, the logarithm of this function (log-likelihood denoted by ℓ) is used:

$$\begin{aligned} \mathcal{L}(M|D) &= P(D|M) \\ \ell(M|D) &= \log P(D|M) \end{aligned} \tag{1}$$

In maximum likelihood estimation (MLE), one is looking for the model (or parameters of the model within a model family) that maximizes the likelihood, or alternatively the log-likelihood of the model given the data which is believed to be the best model describing that data:

$$\begin{aligned} M_{ML} &= \arg\max_{M_i} \mathcal{L}(M_i|D) \\ &= \arg\max_{M_i} \ell(M_i|D) \end{aligned} \tag{2}$$

Some methods that model morphology using maximum-likelihood can be found in [CL02, CL04a].

MAP (Maximum a posteriori): As can be seen from equation (2), in ML estimation method, there is no way of assigning prior probabilities to models. MAP estimation methods select the model based on their posterior probability, i.e., probability of the model after data D is observed. According to Bayes' rule posterior probability denoted by $p(M|D)$ can be defined as:

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)} \tag{3}$$

Here, $P(M)$ is the *prior* probability of the model, i.e., probability of the model before any data is observed. This reflects our belief of model's *correctness* prior to any evidence. Thus, the model that best describes the data can be computed as:

$$\begin{aligned} M_{MAP} &= \arg\max_{M_i} P(M|D) \\ &= \arg\max_{M_i} \frac{P(D|M_i)P(M_i)}{P(D)} \\ &= \arg\max_{M_i} P(D|M_i)P(M_i) \end{aligned} \tag{4}$$

The latter holds because $P(D)$, probability of data, is independent from models and thus does not affect the maximization. [CL05] is one such method.

Authors in [SJ00] mention that most other methods do not use semantics of the input words as a way of unsupervised and knowledge-free morphology induction and rely only on statistics of affixes, thus resulting in problems such as wrong usage of valid suffixes (e.g. wrong segmentation of "ally" into "all" and "y" although "y" is a valid suffix in many other words), failure in identifying morphological relationships in ambiguous cases such as "rating" which might be grouped with "rat" instead of "rate". Their method reads in around 8 million words of random sub-collections of the TREC data-set [VHB97] and computes conflation sets of words (sets of words that are morphologically related) as well as a list of induced rules and list of affixes. Although they seek slightly different goals than we do, they demonstrate that their semantics-aware method helps in morphology induction and rivals Linguistica [Gol01]. They use a *trie* [De 59] for listing potential affixes which can be used to identify words that are likely to have the same stem and thus, are morphologically related. However, identification of such relationships is aided by a semantic similarity measure, since semantically similar words that seemingly have the same stems

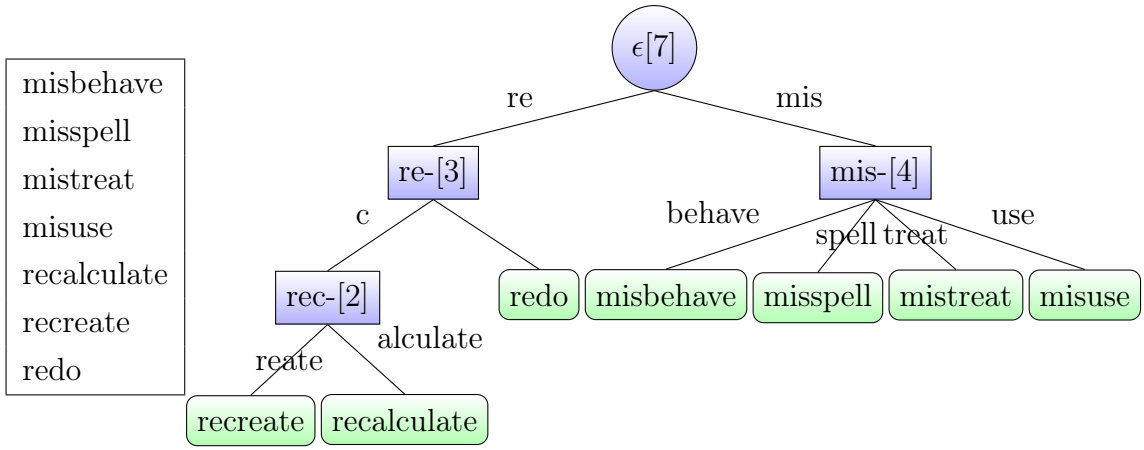

Figure 3: A trie (aka. *digital tree*, *radix tree*, or *prefix tree*) constructed from seven words on the left. The tree illustrated here is a compact trie in which nodes along a path without any branching have been united into one edge. Words are placed on the leaf nodes and the strings inside the middle nodes show the prefix of all words of the corresponding subtree. The number of words in each subtree can be found in brackets in the middle nodes. A high enough frequency for the nodes, provided that they correspond to a long enough string, suggests a potential morphological prefix in the language.

are more likely to be morphologically related. In their work latent semantic analysis (LSA) [LFL98] is used to measure how semantically close two words are. They identify semantic relationship by applying singular value decomposition (SVD) on a term-term matrix, which transforms the words into a k-vector of latent semantic directions. These vectors are then compared using normalized cosine scores (NCS) to acquire information about semantic similarities of words. They conclude that using semantics in knowledge-free induction of morphology can play an important role and frequency-based and semantic-based methods play complementary roles in unsupervised morphology induction [SJ00].

Another related work that takes advantage of semantic information is [BMT02]. Their work is based on the observation that many orthographically similar words are not morphologically related. Most semantically related words are not morphologically related either, but if two words are both orthographically and semantically similar, they are more likely to have the same morphological root. They, however, do not use statistical information such as affix frequency and hope that that this would result in finding rare affixes or words related via non-concatenative morphological processes. Their system is fed a list of words from a corpus for each language and

outputs pairs of words that are found to be morphologically related, but no attempts are made to discover the patterns that relate them. A list of orthographically similar pairs (based on normalized edit distance), and a second list of semantically similar pairs (based on mutual information) is maintained and refined, and finally only the pairs that appear in both lists are reported as morphologically related pairs, ranked by a weighted score based on both similarity measures. Semantic similarities are measured by mutual information criteria computed based on cooccurrences of words in a window. They use an orthographic edit distance for measuring similarities, however, a phonetically inspired measure would also seem to capture useful similarities. In their work, German words that are longer than 9 characters are filtered out since these words end up in the beginning of the list and they do not consider long compounds interesting, however, discovery of stems of compounds specially in languages that employ compounding as one of their fundamental word formation mechanisms (such as German) is an interesting problem. Their system is tested on English and German and they report a high precision in the results and report discovery of several non-concatenative morphological processes such as German plurals formed by umlaut [BMT02].

One of the prior works that is similar to ours both in goal and method, is [AAAK04]. They show how formulation of description length can aid in building an efficient algorithm for morphological segmentation in the following way: the algorithm maintains and refines a dictionary of word forms, by iteratively creating a list of prefix candidates. At each step, the prefix p that achieves the greatest decrease in description length is chosen as the next prefix. Then, all words that have p as a prefix are segmented accordingly. This process is continued until convergence. They test several models of description length including a two-part code approach. They test their method on a large English corpus and a Turkish corpus, however, evaluation is done by visual examination of the first N prefixes.

[Ber08] introduces a method that competed in Morpho Challenge [KCV07]. The method is an unsupervised method that has only three parameters to tune and produces labeled segmentations from a list of input words. Produced segments are labeled as "stem", "prefix", "suffix", or "linking element". It uses the notion of segment predictability, which states that a potential segment boundary could be hypothesized at points where it is difficult to produce the next character. Such predictions are based on transition frequencies between characters in the corpus. Once such potential segments are computed, a list of prefixes and suffixes is computed based on the segments and their frequencies, followed by a list of stems. There are several

other restrictions that are imposed to prevent the model from unwanted behavior, such as elimination of prefixes and suffixes that are only one character long.[8] Frequencies of the segments in the entire corpus are used as a measure for ranking the segments and choosing the best one. This method does not address allomorphy, but addresses some cases of homography, where one segment could be related to more than one stem. The method has been tested as part of the Morpho Challenge 2007 competition on 4 languages [KCV07].

Table 1 summarizes some of the methods which have been devised for addressing learning of morphology.

Name	**Method**	**Notes**
Lingusitica [Gol01]	MDL	
[AAAK04]	MDL, Recursive,	
[BMT02]	Edit distance,Semantics	
Morfessor baseline [CL02]	MDL, HMM, Recursive	
Morfessor FlatCat [GVSK14]	HMM, Semisupervised	
[DN07]	Allomorphy	
[Har55]	LSV	
[Gau99]		
[JM03]	DFA	Identifying hubs in DFA
[KM01]	Supervised, GA, ILP	
[Mon04]		
ParaMor [MCLL09, MLCL08, MLCL04]		
[MEB09]		
Whole Word Morphologizer (WWM)[NF02]	Deterministic	POS-tagged data as input, Identify morphological relationships without morphemes (based on the theory of Whole Word Morphology)

Table 1 – continued on next page

[8] Although this might be a valid thing to do for the four languages that the method has been tested on, it will not work for some other languages such as Russian which have single-character prefixes, besides it builds into the model some a priori non-universal linguistic facts.

Table 1 – continued from previous page

Name	Method	Notes
RePortS [KP06]	Deterministic	Intuitive.
[Dem07]	Deterministic	
[LL10]	Deterministic, Formal Analogy	
[Ber06]	Deterministic	
[SJ00]	Semantics (LSA)	
[SKD02, SKD08]		Developed for learning morphology of Assamese
[SJB02]	Generative probability model	
MetaMorph [TMR10]	Multiple Sequence Alignment (MSA)	

Table 1: Several of previous algorithms developed for morphological segmentation. Unsupervised methods have been highlighted in the table.

4 StateMorph: A MDL Approach to Morphology Learning

In this chapter we describe our method for unsupervised learning of morphology: StateMorph. We try to model the morphology of a language using a finite-state automaton. This can be viewed as a first order Markov chain as well, where the next state is determined only based on the current state and not on the sequence of previous states. For each word, the automaton starts at state S_0 (the initial state), then with some probability, it transits to another state and emits a *morph*. The state that emits the morph is meant to identify which *group* of morphs the emitted morph belongs to. This sequence of transition-emission continues until the whole word is processed. Then the automaton transits to the final state S_F and emits a special *"end of word"* symbol (denoted by #). This process is depicted in Figure 4.

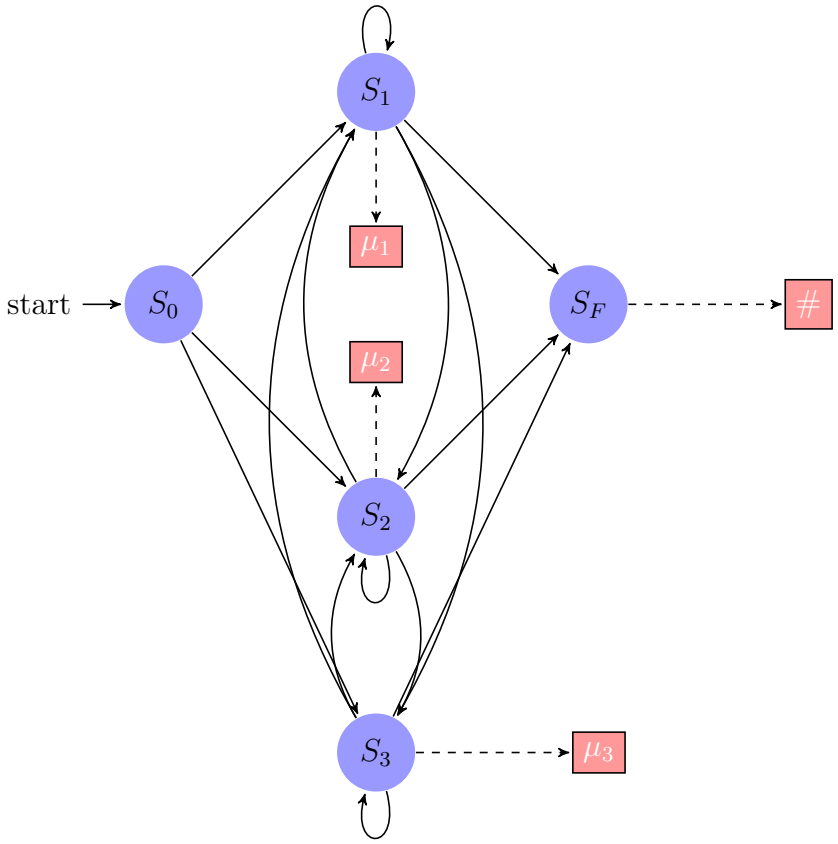

Figure 4: The model as a finite-state automaton with three states. The dashed arrows illustrate emission of strings. S_0 emits nothing and S_F always emits end-of-word symbol #.

The model consists of the following components:

- ***Lexicon***:
  A list of morphs that a state can emit. Each state maintains is own list of emitted morphs.
- ***Transition*** probabilities:
  Probability of each state being the next state, given the current state.
- ***Emission*** probabilities:
  Probability of emitting each morph given the current state.

Ultimately we want each of the states to emit morphs that are categorically the same, for instance a state that emits prefixes, or prefixes with certain characteristics.

This formulation describes the model family that we are going to work with. We now need to choose the model that best describes the data, hence a model selection approach is needed here. As stated by Rissanen's minimum description length principle, the best model to describe a set of data is the one that compresses it the most. So our aim here is to find the model in this model family that can be written down using the smallest number of bits. This raises need for a means of measuring the number of bits required to transmit the model, which will be called the *model cost* or *code-length* here. Model cost for data D consists of 3 components—lexicon cost, transition cost, and emission cost:

$$L(D) = L_{Lex}(D) + L_{Trans}(D) + L_{Emit}(D). \tag{5}$$

For coding the transitions and the emissions, we use the prequential coding method which is introduced next. The formulas for computing the three components of the total cost of the data are given in Section 4.2.

4.1 Prequential Coding

Suppose that we want to construct an efficient code for the data at hand. Such an optimal code could be constructed via a prefix code approach. But constructing an efficient prefix code requires knowledge about the underlying distribution of the symbols in the data. Such a distribution is unknown to us, otherwise there would be nothing to look for and the solution lies in the distribution.

Also one cannot assume that such a real underlying distribution exists. The best one can do, is to estimate a distribution that describes the data, and construct a code based on that. A maximum-likelihood estimation would be a good idea to use here. So the sender can estimate the distribution using the data at hand, construct a code and start transmitting the data using that code. The receiver, on the other hand, needs to be able to *decode* the received message and reconstruct the data. If this condition is not met, we fail to have a proper code. In order for the receiver to properly decode the message, it needs to either know what each symbol in the code stands for, or it should know the distribution of the data that the sender used to construct the code based on. Clearly the receiver does not possess this information unless they are transmitted by the sender, which will add to the code length and make it more cumbersome to compute.

Prequential coding [Daw84] is based on the idea that statistical inference is about

making predictions for future observation based on previous ones, rather than finding the parameters of a distribution that fits the model. Suppose that a sender and a receiver are trying to communicate and the data that needs to be transmitted is a sequence of events $E = \left(e_i\right)_{i=1}^{k}$ where $e_i \in \mathcal{E}$. $\mathcal{E}$ is the finite set of possible event types. Initially, both sender and the receiver start with no information about the distribution of the data that is going to be transmitted, hence they have agreed on a *prior* distribution of the events. They both can construct the same code based on this prior distribution, thus the message will be decodable. The sender then sends the symbol corresponding to e_1, and updates its distribution by adding one e_1 observation. The receiver decodes the message as e_1 and updates its own estimate of distribution of events in the same way. So now once again the sender and receiver have the same distribution, and hence share the same code. This process goes on until the whole sequence has been transmitted. This way, both sides will maintain the same distribution at every step. It can be shown that [Wet13, Hil12] using Bayesian marginal likelihood, one can transmit the events in E using a code the length of which is equal to

$$\begin{aligned} L_{prequential}(E) = &- \sum_{i=1}^{N} \log \Gamma(c_i + \alpha_i) + \sum_{i=1}^{N} \log \Gamma(\alpha_i) \\ &+ \log \Gamma \left[\sum_{i=1}^{N} (c_i + \alpha_i) \right] - \log \Gamma \left[\sum_{i=1}^{N} (\alpha_i) \right] \end{aligned} \tag{6}$$

where $N = |\mathcal{E}|$, c_i is the number of times event $x_i \in \mathcal{E}$ is observed in the whole data, and α_i is the prior count of x_i. $\log x$ represents the binary logarithm of x throughout the thesis. Γ stands for the gamma function. If our counts c_i and priors α_i are integers, the gamma function can be rewritten in terms of factorials since $\Gamma(n) = (n-1)!$ for any positive integer $n \in \mathbb{N}$. In case of non-integer priors, one must follow the general definition of Γ

$$\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx.$$

In our work we use uniform priors ($\alpha_i = 1$), thus the code-length can have a simpler form of

$$L_{prequential}(E) = -\sum_{i=1}^{N} \log \Gamma(c_i + 1) + \log \Gamma\left(\sum_{i=1}^{N} c_i + N\right) - \log \Gamma(N). \tag{7}$$

Equation (7) gives a formula for computing the code length based on only the frequency of events and the priors which makes it suitable for being used in the optimization process.

4.2 Computing the Code Lengths

Here we describe how each of the three components of the model are coded and we give the formulas for computing the respective costs.

- **Lexicon** code-length: There are several alternatives for coding the lexicon. The aim here is to efficiently encode a list of morphs which are emitted from states. One method to do this, used also in [CL02], is to assume uniform distribution on the symbols and use a fixed-length code for each symbol. The number of bits required for morph μ is then equal to

$$L(\mu) = (|\mu| + 1) \cdot \log(|\Sigma| + 1) \tag{8}$$

where Σ is the augmented alphabet: the set of all symbols used in the language plus a special end of word symbol ("#") and $|\mu|$ is the number of symbols in morph μ. As a whole, the lexicon then can be represented in

$$L_{Lex} = \sum_{i=1}^{K} \left[\Big(\sum_{\mu \in Lex_i} L(\mu) \Big) + \log(|\Sigma| + 1) \right] \tag{9}$$

bits where Lex_i represents all morphs that are emitted at least once for state S_i and K is a pre-determined number of states. We need to encode the lexicon in a way that the receiver can determine when one state ends and the next one begins. Transmission of an empty morph at the end of each state does this, which costs $\log(|\Sigma| + 1)$ bits. Note that since the start state does not emit anything and the end state unconditionally emits the end of word symbol the cost of lexicon for them is zero since nothing needs to be coded.

Bearing in mind that when coding the lexicon for the states, the order in which they appear is not relevant since we are just coding a set, we can more efficiently code the lexicon using

$$L_{Lex} = \sum_{i=1}^{K} \left[\sum_{\mu \in Lex_i} L(\mu) + \log(|\Sigma| + 1) - \log |Lex_i|! \right] \tag{10}$$

bits. The $\log |Lex_i|!$ in code-length is saved due to the fact that we do not need to transmit the lexicon in any specific order, thus the sender is free to choose the order in which the morphs are transmitted. Since there are $|Lex_i|!$ permutations of the morphs, we can save $\log |Lex_i|!$ bits.

Transitions and emissions are coded prequentially. To use prequential coding, we need to formulate the data as a sequence of events. The events here are transitions between the states and emission of morphs from the state since each segmentation is a sequence of transitions and emissions. Here we describe the event space for transitions and emissions.

- **Transitions** code-length:
  A transition from state S_i to state S_j can be considered as an event. The event space then can be defined as $\mathcal{E}_{Trans} = \{ (i, j) \mid 0 \leq i, j \leq N \}$ and transitions can then be coded prequentially.

- **Emissions** code-length:
  Similarly, once the set of possible morphs (lexicon) is known—i.e., has been coded—it can serve as our event set for prequential coding and each emission from a given state S_i can be an occurrence of the corresponding event: $\mathcal{E}_{Emit} = Lex_i$.

Once the code length function $L(D)$ is defined as in Equation (5), we can use it as the objective function which the learning algorithm will minimize, using a search procedure, which we describe next.

4.3 Search Algorithm

Given a list of words, we want to search for the best segmentations for the words, i.e., the segmentations that minimize the code-length.

We use an approach based on expectation-maximization (EM) [DLR77]. Expectation-maximization is a method for estimating the maximum likelihood (ML) or maximum a posteriori (MAP) parameters of a statistical model where the model has missing data or latent variables. It consists of two steps: expectation (E) and maximization (M).

In the E step, the unobserved data are estimated based on the current parameters of the model. The estimated data along with the observed data form the complete data, which are fed to the M step which will find the ML (MAP) parameters in order to calculate new parameters for the model. The algorithm then alternates between these two steps which guarantees that the objective improves at each iteration.

As listed in Algorithm 1, we start by a completely random segmentation of words and iteratively search for better segmentations given the rest of the data (the E step). Once we have a set of new segmentations, we can build the model based on that (the M step).

Searching for new segmentation is done using a dynamic programming algorithm that is explained in detail in Section 4.4. A *classification* for a word segmentation is simply a list of states that each segment is emitted from. This is called classification since we use the term *class* for states as they are meant to correspond to classes of morphs.

Algorithm 1 StateMorph Search algorithm

```
1:  procedure StateMorph-Search
2:      for all word w ∈ W do
3:          S_i ← Generate a random segmentation
4:          C_i ← Generate a random classification for S
5:          Register S_r and C_r in the model by updating the event counts.
6:      end for
7:      repeat
8:          for w_i ∈ W do
9:              Deregister current classification and segmentation of w_i
10:             (S_i, C_i) ← Search for a new segmentation and classification for w_i
11:             Register S_i and C_i in the model
12:         end for
13:     until convergence is achieved
14: end procedure
```

It is important to note that since we only compute one segmentation for each word, our model is following the so-called *hard* EM approach. To use the *soft* EM, one can compute the probability of all possible segmentations and classifications for each word and adjust the EM algorithm accordingly. This is computationally more demanding and is left as part of the future work plan.

4.4 Segmentation of Words by Dynamic Programming

One crucial part of the algorithm is searching for the best segmentation for a word, given segmentations for the rest of the words in the word list. We use dynamic programming for this purpose, similar to the approach we have previously used in other works for analysis of etymological data [WNRY12, WNRY13, NY14] and transliteration generation [NPY13].

For a word of length n, there are 2^{n-1} different segmentations since there are $n-1$ potential split points. Furthermore, for each of these segmentations, depending on the number of segments and number of states in the system (K), there is a finite, but generally large number of different classifications.

If a word is split into s segments, assuming that there are K classes, because each of the segments can be emitted from any of the classes, there will be K^s possible ways to classify it. On the other hand, there are $\binom{n-1}{s-1}$ different segmentations of length s, thus, the number of different segmentations of length s along with classifications, denoted by $N(s)$ can be calculated as

$$N(s) = \binom{n-1}{s-1} K^s \tag{11}$$

and since a word of length n can be segmented into at least 1 and at most n segments (empty segments are not allowed), the total number of possible segmentations and classifications for a word is equal to

$$\begin{aligned} N(w) &= \sum_{s=1}^{n} N(s) \\ &= \sum_{s=1}^{n} \left[\binom{n-1}{s-1} K^s \right] = K(K+1)^{n-1}. \end{aligned} \tag{12}$$

Thus, the number of possible solutions for a word of length n given that the system has K states is exponential, which makes it intractable for an exhaustive search,

bearing in mind also that there is generally a large number of words in the data set.[9]

Dynamic programming is a bottom-up approach to problem solving which works by taking into account *optimal substructures* and *overlapping subproblems*. A problem is said to have optimal substructure if the optimal solution to it can be constructed from optimal solution(s) to its subproblems. If subproblems are repeated as parts of many candidate solutions, the problem is said to exhibit overlapping subproblems. This can be seen in some recursive algorithms where a call to solve a subproblem is repeated over and over.

If a problems exhibits both of these characteristics, dynamic programming can be used to solve it in considerably less time compared to the recursive approach. Time saving is done via avoiding recalculation of solutions to the subproblems.

Let us start by defining how we break the complex problem into smaller subproblems.

For a word w, we denote by σ_a^b a **substring** of w from position a to position b inclusive. Indices here are in the range between 1 and n where n is the length of w.

We will write σ_1^b as σ^b for simplicity. This is the **prefix** of w up to b. Similarly a **suffix** of w like σ_a^n can be written as σ_a. This way, σ^n will be equivalent to the whole word w. Since we add a special *end of word* symbol to all words, let us denote the augmented word by $\sigma^{n+1} = w\#$. Similarly, a special substring σ^0 represents the empty string ϵ.

Let us denote cost of segmenting σ^i and being at state S_j with $C(\sigma^i, S_j)$. We are after the solution which minimizes $C(\sigma^{n+1}, S_F)$ (i.e., segmenting the whole augmented word and ending up in the final state).

By definition, the final state S_F can only emit the end-of-word symbol #. Therefore the whole word σ^n must have been emitted before the system can transit to S_F. Assuming that we have solved the problem of segmenting σ^n and arriving at a particular state and thus know the best way to do so, we simply enumerate transition from each of those states to S_F and emitting # there, and choose the option that minimizes the cost. This is formulated as

$$C(\sigma^{n+1}, S_F) = \min_{0<l\leq K} \Big\{ C(\sigma^n, S_l) + C_{Trans}(S_l, S_F) + C_{Emit}(\#|S_F) \Big\}. \qquad (13)$$

[9]Since we want to model the whole language, ideally we would like to have all possible words of the language. This of course is impossible because of generative nature of language, but we still want to work with the longest list of words possible.

Similarly we can define the solution of segmenting a prefix and ending up in a particular state in terms of segmentation of shorter prefixes. The number of different ways to segment σ^i and end up in state S_j can be enumerated as different ways to segment a shorter prefix σ^k and end up in a state S_l, then transit from S_l to S_j and emit the rest of the prefix (σ^i_{k+1}). Since we are interested in the best (least expensive) way to do this, we minimize over these possible ways. This recursive relation is formulated as

$$C(\sigma^i, S_j) = \min_{\substack{0<k<i \\ 0\leq l\leq K}} \Big\{ \overbrace{C(\sigma^k, S_l)}^{\text{Cost of segmentation up to } k} + \underbrace{C_{Trans}(S_l, S_j)}_{\text{Transit from } S_l \text{ to } S_j} + \overbrace{C_{Emit}(\sigma^i_{k+1}|S_j)}^{\text{Emitting substring } \sigma^i_{k+1} \text{ from } S_j} \Big\}. \quad (14)$$

This formulation assumes that the problem has optimal substructures, meaning that for a word, how we have segmented σ^k and ended up in S_l does not affect the cost of later transitions and emissions, therefore these parts of the final cost can be minimized independently. In reality, however, segmentation of the substring σ^k might effect the cost of later events in the dynamic programming matrix of the word. For example introducing a new morph to a state S_k will make it cheaper to emit the same morph from the same state later in the word if the rest of the word also contains that morph. This formulation will assign the same cost to both instances of the emission even though the second emission should be cheaper. This might change the optimal solution to the word segmentation, however, it is not very common to have the same morph more than once in a word, thus this should not have a considerable effect on the overall method. If the input data is large and one word at a time is resegmented, optimal substructures could be assumed in order to solve the problem in tractable time. Otherwise one will not be able to address the problem in this way with tractable time complexity.

We now need to define $C_{Trans}(S_x, S_y)$—cost of transition from state S_x to S_y— and $C_{Emit}(\mu|S_x)$—cost of emitting morph μ from state S_x. These functions quantify the difference in total cost of the data if occurrence of event e is added to the data. This can be computed by calculating the difference in the prequential cost. Using the formula for prequential code-length in Equation (7) we have

$$\Delta L_{prequential} = L_{prequential}\big(E \cup \{e\}\big) - L_{prequential}(E)$$

$$
\begin{aligned}
&= -\sum_{\substack{i=1\\ i\neq x}}^{N} \log\Gamma(c_i+1) - \log\Gamma(c_x+2) + \log\Gamma\Big(\sum_{i=1}^{N} c_i + N + 1\Big) - \log\Gamma(N) \\
&\quad - \Bigg[-\sum_{i=1}^{N} \log\Gamma(c_i+1) + \log\Gamma\Big(\sum_{i=1}^{N} c_i + N\Big) - \log\Gamma(N) \Bigg] \\
&= -\log\Gamma(c_x+2) + \log\Gamma(c_x+1) \\
&\quad + \log\Gamma\Big(\sum_{i=1}^{N} c_i + N + 1\Big) - \log\Gamma\Big(\sum_{i=1}^{N} c_i + N\Big) \\
&= -\log\frac{\Gamma(c_x+2)}{\Gamma(c_x+1)} + \log\frac{\Gamma\big(\sum_{i=1}^{N} c_i + N + 1\big)}{\Gamma\big(\sum_{i=1}^{N} c_i + N\big)} \\
&= -\log(c_x+1) + \log\Big(\sum_{i=1}^{N} c_i + N\Big) \\
&= -\log\frac{c_x+1}{\sum_{i=1}^{N} c_i + N}
\end{aligned}
\tag{15}
$$

where x is the index of the event that e belongs to. This is in case e is already in the event set. If $e \notin \mathcal{E}$, $c_x = 1$ and we have to compute the corresponding cost separately using

$$
\begin{aligned}
\Delta L_{prequential} &= L_{prequential}(E \cup \{e\}) - L_{prequential}(E) \\
&= -\sum_{i=1}^{N} \log\Gamma\big(c_i+1\big) - \log\Gamma\big(c_x+1\big) \\
&\quad + \log\Gamma\Big(\sum_{i=1}^{N} c_i + c_x + N + 1\Big) - \log\Gamma\big(N+1\big) \\
&\quad - \Bigg[-\sum_{i=1}^{N} \log\Gamma(c_i+1) + \log\Gamma\Big(\sum_{i=1}^{N} c_i + N\Big) - \log\Gamma(N) \Bigg] \\
&= -\log\Gamma(2) + \log\Gamma\Big(\sum_{i=1}^{N} c_i + N + 2\Big) - \log\Gamma\Big(\sum_{i=1}^{N} c_i + N\Big) \\
&\quad - \log\Gamma\big(N+1\big) + \log\Gamma\big(N\big) \\
&= -\log\frac{\Gamma\Big(\sum_{i=1}^{N} c_i + N\Big)}{\Gamma\Big(\sum_{i=1}^{N} c_i + N + 2\Big)} - \log\frac{\Gamma\big(N+1\big)}{\Gamma\big(N\big)} \\
&= -\log\frac{\Gamma\Big(\sum_{i=1}^{N} c_i + N\Big)}{\big(\sum_{i=1}^{N} c_i + N + 1\big)\big(\sum_{i=1}^{N} c_i + N\big)\Gamma\Big(\sum_{i=1}^{N} c_i + N\Big)} - \log N
\end{aligned}
$$

$$= -\log \frac{N}{\big(\sum_{i=1}^{N} c_i + N\big)\big(\sum_{i=1}^{N} c_i + N + 1\big)}. \tag{16}$$

This holds since $\log \Gamma\big(2\big) = 0$ and $\Gamma(t+1) = t\Gamma(t)$ for all t.

Based on these formulas, we can calculate the code-length for transitions among states using

$$C_{Trans}(S_x, S_y) = -\log \frac{f_{Trans}(x,y) + 1}{\sum_{i=0}^{K-1} \sum_{j=1}^{K} \big(f_{Trans}(i,j) + 1\big)} \tag{17}$$

where $f_{Trans}(x,y)$ is the number of transitions from S_x to S_y.

For emission of morph μ from state S_x we have two cases:

$$C_{Emit}(\mu|S_x) = \begin{cases} -\log \frac{f_x(\mu)+1}{f(S_x)+|S_x|} & \mu \in S_x \\ -\log \frac{|S_x|}{\big(f(S_x)+|S_x|\big)\big(f(S_x)+|S_x|+1\big)} + L(\mu) & \mu \notin S_x \end{cases} \tag{18}$$

where $f_x(\mu)$ is the number of times μ is emitted from state, $f(S_x) = \sum_{\nu \in S_x} f_x(\nu)$, and $|S_x|$ is the number of distinct morphs emitted from state S_x. In the second case, if $\mu \notin S_x$, we need to add it to the lexicon which will cost $L(\mu)$ bits.

Now that we have defined the recursive formulae for finding the best solution, we can formulate them in terms of a dynamic programming matrix M depicted in Figure 5. Each cell M_{ji} of the matrix is defined as

$$\mathbf{M}_{[K+2]\times[n+2]} := [m_{ji} = C(\sigma^i, S_j)]. \tag{19}$$

The problem is now reduced to filling in this dynamic programming matrix. The algorithm to do so is explained here.

Since state S_0 is a special *start* state and does not emit anything, or in other words emits ϵ unconditionally, elements of the first row of the matrix which correspond to a non-empty string are not defined, and we can define a cost of infinity ($+\infty$) for them:

1: **for** i from 1 to n **do**
2: $\quad M_{0i} \leftarrow +\infty$
3: **end for**

And because we start at S_0 with empty string ϵ, the cost of this cell is defined to be 0:

	ϵ	σ^1	σ^2	...	σ^j	...	σ^n	#
S_0	0	∞	∞	∞	∞	∞	∞	∞
S_1	∞							∞
S_2	∞							∞
...	∞							∞
S_i	∞				$C(\sigma^j, S_i)$			∞
...	∞							∞
S_K	∞							∞
S_F	∞	∞	∞	∞	∞	∞	∞	$L(w)$

Figure 5: Dynamic programming matrix for a word of length n. Rows of the matrix correspond to the states of the model and columns correspond to the prefix of the word. Each cell holds the cost of segmentation of the prefix and arriving at the corresponding state.

4: $M_{00} \leftarrow 0$

Also in states other than S_0, we must have emitted some part of the word, so the remaining cells in the first column have infinite cost:

5: **for** j from 1 to $K + 1$ **do**
6: $\quad M_{j0} \leftarrow +\infty$
7: **end for**

Similarly, by definition, when we are in state $S_F = S_{K+1}$, the whole augmented word must have been emitted, thus the last row of the matrix is again undefined, except for the bottom-right element which will hold the cost of the final solution:

8: **for** i from 0 to n **do**
9: $\quad M_{K+1,i} \leftarrow +\infty$
10: **end for**

Also only the final state S_{K+1} can have emitted the whole augmented word and thus the last column except for the bottom-right element must be equal to $+\infty$:

11: **for** j from 0 to K **do**
12: $\quad M_{j,n+1} \leftarrow +\infty$
13: **end for**

Now we can start filling in the remaining matrix elements one column at a time starting from left. This way, the elements of the matrix are calculated before they

are needed by later matrix elements:

14: **for** i from 1 to n **do**
15: **for** j from 1 to K **do** ▷ Compute element M_{ji}
16: Candidates $\leftarrow \{\}$
17: **for** k from 0 to $i-1$ **do**
18: **for** l from 0 to K **do**
19: candidate_cost $\leftarrow M_{lk} + C_{Trans}(S_l, S_j) + C_{Emit}(\sigma^i_{k+1}|S_j)$
20: Append $\langle l, k, \text{candidate\_cost}\rangle$ to Candidates
21: **end for**
22: **end for**
23: $\langle state, prefix, cost\rangle \leftarrow \arg\min_{\langle l,s,c\rangle \in \text{Candidates}}\{c\}$ ▷ The best candidate
24: $M_{ji} \leftarrow cost$
25: $parent(j,i) \leftarrow (state, prefix)$ ▷ For obtaining the final segmentation
26: **end for**
27: **end for**

In line 20 of the algorithm we keep a list of possible candidates. For a completely greedy search we could have stored only the solution that minimizes the cost instead of a list. However, since later we will use simulated annealing for avoiding local optima where the minimum solution will not necessarily be chosen, we keep a list for further processing. The simulated annealing approach is explained in Section 4.5.

In line 25 we store the parent cell—the cell from which the transition is made—of the current cell. This will be useful for reconstructing the path and obtaining the final segmentation of the word in the later steps.

Now the only thing to do is to compute the final cell $M_{[K+1][n+1]}$ which is computed in a slightly different way than others as defined in Equation (13).

28: Final Candidates $\leftarrow \{\}$
29: **for** l from 1 to K **do**
30: candidate_cost $\leftarrow M_{ln} + C_{Trans}(S_l, S_F)$
31: Append $\langle l, n, \text{candidate\_cost}\rangle$ to Final Candidates
32: **end for**

And now choose the best one

33: $\langle state, prefix, cost\rangle \leftarrow \arg\min_{\langle l,s,c\rangle \in \text{Final Candidates}}\{c\}$ ▷ The best candidate
34: $M_{[K+1][n+1]} \leftarrow cost$
35: $parent(K+1, n+1) \leftarrow (state, prefix)$

The cost of best segmentation for the word is now stored in $M_{[K+1][n+1]}$. To obtain the segmentation and classification for the word, one just needs to reconstruct the path that produced the minimum cost. Information stored via the *parent* operator is useful here. The algorithm for path reconstruction can be found in Algorithm 2.

Algorithm 2 Reconstruction of path

```
1: function RECONSTRUCT-PATH(M)
2:     Segments ← ⟨#⟩
3:     Classification ← ⟨K + 1⟩                                   ▷ Initialize
4:     current state ← K + 1          ▷ Start from the last filled matrix element
5:     current position ← n + 1
6:     p ← current position
7:     while current position ≠ NIL and current state ≠ NIL do
8:         ⟨current state, current position⟩ ← parent(current state, current position)
9:         Prepend current state to Classification
10:        Prepend σ^{p−1}_{current position} to Segments
11:        p ← current position
12:    end while
13:    return Segments, Classification
14: end function
```

4.5 Local Optima and Simulated Annealing

Due to the greedy nature of the algorithms explained in Section 4.4 and since the model starts with completely random initial segmentations, the search algorithm is very vulnerable to converging to a local optimum. In fact, the only thing the converged model depends on is the initial segmentation and classification of the words.

This is a potential obstacle in global optimization problems for which no closed form solutions are available and the search space is too large for an exhaustive search. In this kind of situations, if the goal can be regarded as finding a sufficiently good solution in a reasonable time rather than the best possible solution, several metaheuristic methods could be employed. One such metaheuristic is *simulated annealing* which works by deploying similarities between statistical mechanics and combinatorial optimization.

Simulated annealing has been explained in [KGV83] and [Čer85]. It avoids local

optima by allowing acceptance of worse solutions while searching for the sufficiently good one. Probability of accepting a worse solution decreases over time, typically in a way that in the beginning the solutions are mostly random and at the final stages of the search a completely greedy approach is followed and no worse solutions are accepted. This probability of accepting worse solutions is inspired by the concept of *slow cooling*. The idea in simulated annealing is that when optimizing for finding a certain solution, instead of always choosing the best succeeding solution, with some probability which depends on the current *temperature* of the model, we choose a worse solution. As the temperature drops, this probability fades away and at final stages the algorithm is equivalent to a greedy search.

The greedy search for the best segmentation for the words in the corpus as presented above quickly converges to local—far from global—optima. To avoid local optima, the simulated annealing approach is followed, with temperature T varying between fixed starting and ending values, T_0 and T_F, and a geometric cooling schedule, α. In Equation 13, rather than using "min" operator to determine the best cell from which to jump to the given cell m_{ij} in the DP matrix, we proceed as follows: Each candidate cell m_{qb} in the matrix is a pair (state, symbol-position). We sort all of the $(j-1) \times K + 1$ candidate cells for m_{ij}, where $j-1$ is the number of preceding columns, by their accumulated cost. We subtract the cost of the cell with smallest cost c^* (the best solution for (q, b)) from the rest of the entries in the sorted list, to get a list $\{d_i\}$ of cost *differences*; now, $d_0 = 0 \leq d_1 \leq d_2 \leq ...$ The d_i's are non-negative and increasing.

We divide the list through by the temperature T, to get a list $\{\bar{d}_i\}$. When T is large, the numbers $\bar{d}_i$ are all small; as T cools, $\bar{d}_i$ become very large.

For each $\bar{d}_i$, we let $\hat{d}_i = e^{-\bar{d}_i}$, to get a list $\{\hat{d}_i\}$. Now, $\hat{d}_0 = 1$, and the rest of the $\{\hat{d}_i\}$ are positive and non-increasing between 0 and 1. When T is large, the numbers $\hat{d}_i$ are all close to 1, and as T cools, all $\hat{d}_i$ become close to zero. These values are defined as

$$\begin{aligned} c_i &:= \{c | \langle l, s, c\rangle \in \text{Candidates}\} \\ c^* &= \min_i \{c_i\} \\ \hat{d}_i &:= e^{-\frac{c_i - c^*}{T}} = e^{-\frac{d_i}{T}} = e^{-\bar{d}_i}. \end{aligned}$$

We next generate a random number r in $[0, 1)$, and define the simulated annealing window to be $[r, 1]$. We choose the smallest entry $\hat{d}_+$ from the list that is still within

the window, i.e., the **smallest** number for which $r \leq \hat{d}_+ = e^{-\bar{d}_+}$. Usually in the simulated annealing process, a random candidate from the window would be selected. However it would require generation of another random number for each cell for each word per iteration which increases the running time of the algorithm. We have experimented with both selection methods and in our experiments no meaningful differences are visible in their performance.

The process of choosing the solution (a parent cell for cell X) at each iteration for a given word follows this algorithm:

- Sort all candidate cells by their cost, ascending.
- $c^* \leftarrow$ first element in sorted list (This is the best candidate; the one that would have been chosen in a a greedy search).
- For each candidate cell with cost c in the list: compute $\hat{d} = e^{-\frac{c^* - c}{T}}$.
- Generate a random number r in $[0, 1)$.
- Choose the smallest item in the list where $r \leq \hat{d}$.

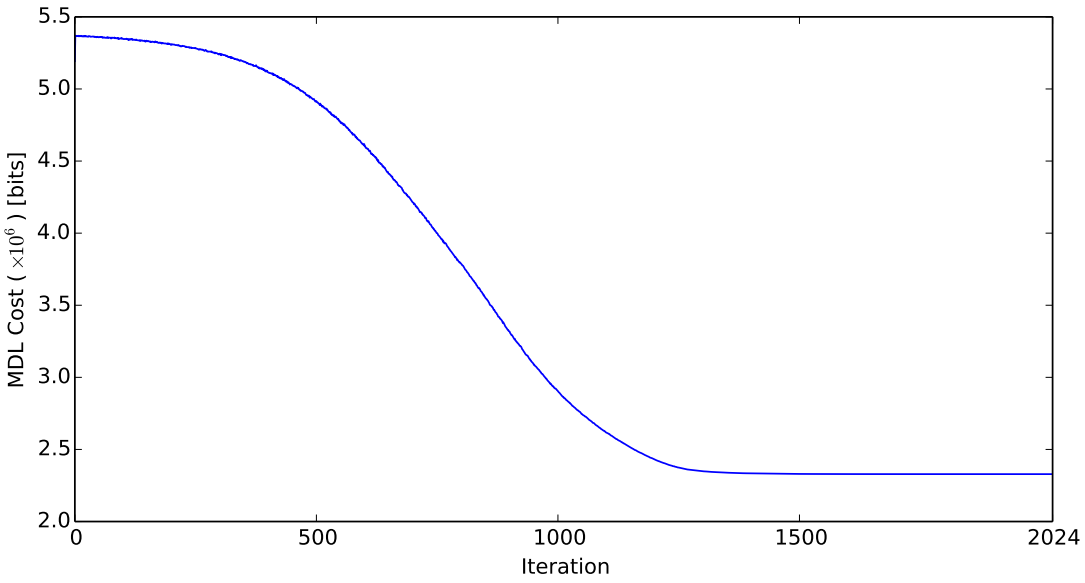

Figure 6: MDL cost in a sample run under simulated annealing

This ensures that the model does not always greedily choose the best solution, and enables it to initially take random jumps to avoid local optima. The learning algorithm is run to convergence on the corpus. An example of the curve of the MDL cost for Finnish is in Figure 6. The initial plateau shows the "burn-in" phase of the

learning. As can be seen, the model cost does not have steep drops and it rather decreases more gradually.

4.6 Parallelization

Using simulated annealing with the chosen parameters requires a large number of iterations to converge. Since this is done in an iterative way and we usually have a relatively long list of words as the input data, it takes a long time for one run of the algorithm to converge. The loop in lines 7-13 of Algorithm 1 is one iteration of the algorithm. Each iteration i requires the previous iteration $i-1$ to have finished so that the new model (lexicon and transitions) are ready to be used for calculating required costs. Inside each iteration, however, the words can be resegmented in parallel. We divide the words into n chunks of roughly the same size and process them in parallel on different machines to reduce the running time. Several approaches to parallelization of the algorithm have been tried in the course of this thesis work but only the final approach is discussed here.

4.6.1 MapReduce Programming Model

MapReduce is a programming model which is designed for processing large data-sets on clusters of machines [DG08, Läm08]. For addressing problems in this style, input and output data are represented in a form of key/value pairs. One needs to define two fundamental operations: *Map* operation which produces a set of intermediate key/value pairs from the input pairs. The MapReduce library then groups all pairs with the same key. These values are then sent to the *Reduce* operator. This operator will reduce a key and all values associated with it (merged after map) to a single (or sometimes zero) key/value pair. It has been shown that many problems can be formulated to work with this programming model [DG08]. MapReduce helps users save time compared to iterative runs since it can run map and reduce operations in parallel. Figure 7 shows an illustration of MapReduce work flow. There are several implementations of MapReduce such as Apache Hadoop [Whi09], Spark [ZCF+10], Disco [MTF11] among others.

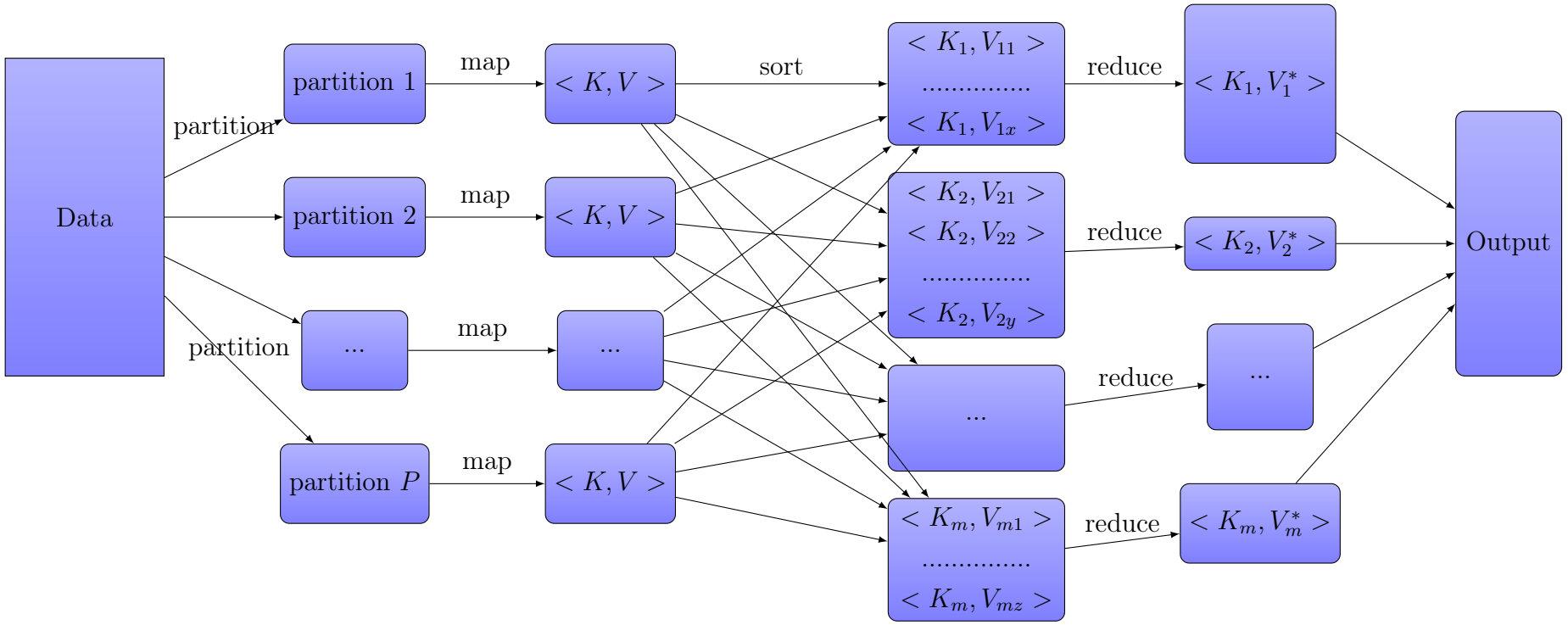

Figure 7: MapReduce programming model. Operations for each step are demonstrated. At the initial step, the input data is split into several partitions, each data item in the partitions is then **map**ped to a key and value (K, V) pair. These pairs are then sorted so that pairs with the same key are in the same group. The resulting groups then undergo the **reduce** operation and the results are collected after that.

In this thesis work, we can define the *map* operation to be re-segmenting a word using the fixed model, and *reduce* operation can be defined as calculating the model for next iteration based on segmentations of all the words.

4.6.2 Iterative MapReduce and *Twister*

Most implementations of MapReduce are suitable for single problems that can be expressed in MapReduce formulation, but they do not necessarily support iterative MapReduce. In our case, after each iteration ends, segmentations of words are reduced into a model, which is then used by the next iteration which is another instance of MapReduce. In most implementations, the workers (processes that do the map tasks) release all the resources that they have acquired right after they finish the tasks, however, to save time, we need the workers to do the same task with the new model. Since there are usually a high number of iterations, if the time required for initializing an instance of the resegmenter (worker) is high, then the overall runtime of the program will not be less than the sequential version and no time is saved.

Twister is an implementation of MapReduce which supports iterative MapReduce and is suitable for iterative and long-running MapReduce tasks [ELZ+10]. Similar to other MapReduce implementations, the library requires the user to provide several operations, such as *initialize*, *map*, *reduce*, etc. An illustration of how the program works using twister is shown in Figure 8.

The possibility to use the implementation of the search algorithm in a cluster is provided using Twister, so the user can decide to whether run the program sequentially on one machine, or use a cluster of machines and choose the number of parallel workers. The input words are then divided among the workers as evenly as possible and the workers are initialized. At the beginning of each iteration, the master sends the model to the workers and asks them to re-segment their words based on the model. Each worker then builds their own local model based on the new segmentations. The reduce task is to accumulate these *local* models into one model which is received by the master and redistributed again for the next iteration. One important point which helps save time in the process is that the master does not need to receive the full segmentation for each word at each iteration. Only transitions and emissions based on the list of words is enough. Only once convergence is reached, master will need access to the full segmentations for purpose of logging the final results.

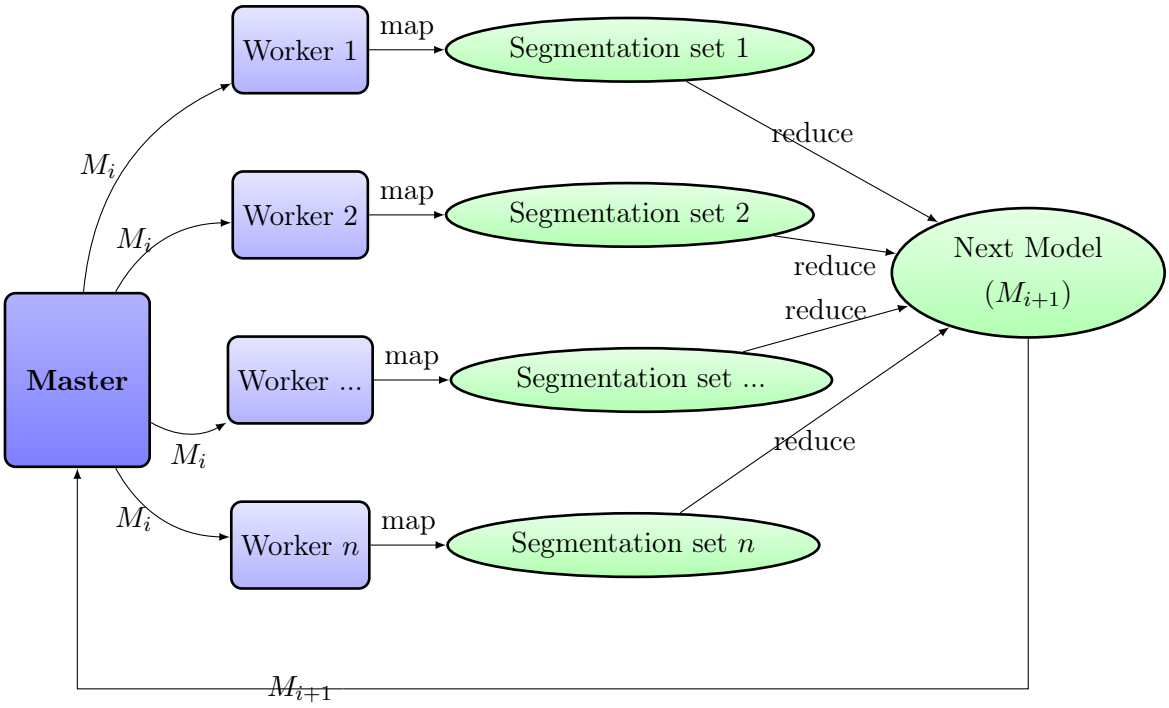

Figure 8: Illustration of resegmentation using *Twister*. The *master* process initializes a number of workers (pre-determined by the user based on the available resources) and assigns a set of words to each of them as evenly as possible. Then, for each iteration i the master broadcasts the current state of the model M_i (lexicon and transitions) to the workers and asks them to resegment their assigned words in parallel, based on M_i. The results are then reduced into a single model M_{i+1} which will be used in the next iteration.

5 Evaluation of Segmentations

Evaluation of methods for unsupervised learning of morphology is a complicated task. Depending on the goal the method is seeking, the evaluation is different; for instance a system that is performing full morphological analysis requires a different evaluation method than one that outputs pairs of morphologically related words.

Since in this work we are interested in morphological segmentation of words, we will focus on evaluation methods of such outputs. This chapter is organized as follows: after a brief introduction, an overview of prior work in evaluation is presented, followed by a discussion of need for a new evaluation method. Finally our proposed method is explained starting in Section 5.5. This evaluation method is published in [NY16].

5.1 Quantitative vs. Qualitative Evaluation

One way of observing how the system is performing is by visual inspection of the output. Although it is understood that this does not give a rigorous measure of performance that could be used for comparison across methods, it still gives an indication of the performance of the methods compared to the expected behavior. Visual inspection of segmentations and/or discovered morphological patterns and comparing them to linguistic knowledge is something that we also report for our work. Although there are methods that only rely on qualitative evaluation, it should be understood that quantitative measures of performance should be preferred over qualitative measures. Quantitative methods, if devised carefully, are in general more objective and give rigorous measures of how methods are performing compared to each other.

5.2 Direct vs. Indirect Evaluation

Since many methods of unsupervised morphology learning are motivated by using them as part of larger systems that perform other tasks such as parsing, machine translation, speech recognition, information retrieval, etc., one way to evaluate the system is by measuring the performance of the larger system that has the morphological learning method as its sub-system [VTS+11]. Measuring how much the use of the to-be-measured morphological unit helps in performing the task of course gives an indication of how good the system is, it should be noted that this method of evaluation complicates the tasks since the measured performance depends on the overall approach used for the larger task.

Methods that measure the performance of the tools only based on their output without involving any other task are called the *direct* evaluation methods. The evaluation algorithm proposed in this chapter also falls in this category.

5.3 Existing Evaluation Methods

A review of evaluation methods for different morphological learning tasks can be found in [VTS+11], which also introduces several new variations of the existing algorithms. Beyond morphological segmentation, these tasks include *clustering of word forms* and *full morphological analysis*.

Direct evaluation methods are, in general, believed to better reflect the characteris-

tics of the algorithm, while indirect methods complicate the evaluation task, since one needs to minimize the effect of other factors in the larger task [VTS+11].

Other evaluation methods, which evaluate more than segmentations, or require more output from the model than only morphological segmentations, are described in [SM10, VTS+11].

A widely used approach [KCL06, SB08, PCT09] for evaluating segmentation of words is the boundary *precision*, *recall*, and *accuracy* (BPRA[10]), based on the number of reference boundaries found and missed by the learning algorithm. Suppose that a set of *correct* segmentation points for the given words could be provided. We call these the *reference* or *gold standard* segmentations. The segmentation task could then be viewed as an information retrieval (IR) task, where the items to be retrieved are the segmentation boundaries. Then the standard IR evaluation measures precision(P), recall (R), and F-score (F) can be defined as:

$$\begin{aligned} P &= \frac{|\{\text{gold standard boundaries}\} \cap \{\text{retrieved boundaries}\}|}{|\{\text{retrieved boundaries}\}|} \\ R &= \frac{|\{\text{gold standard boundaries}\} \cap \{\text{retrieved boundaries}\}|}{|\{\text{gold standard boundaries}\}|} \\ F &= 2 \cdot \frac{P \cdot R}{P + R} \end{aligned} \tag{20}$$

Alternatively, using notation from binary classification, precision, recall, F-score and accuracy (A) can be computed as:

$$\begin{aligned} P &= \frac{tp}{tp + fp} \\ R &= \frac{tp}{tp + fn} \\ A &= \frac{tp + tn}{tp + tn + fp + fn} \end{aligned} \tag{21}$$

where tp, tn, fp, fn denote number of true positives, true negatives, false positives and false negatives, respectively. Positive (negative) means that the model found (did not find) a morph boundary annotated in the gold standard.

One evaluation scheme, based on the BPRA method and related to our work, can be found in [CL04b, CLLV05]. To allow the model to choose one of several alternative reference segmentations, they define *fuzzy* boundaries; instead of a strict segmentation points in words, the model will receive credit if the segmentation boundary

[10]Most earlier works do not include accuracy as one of the measures, however we will include it in this paper along with precision, recall, and F-score since it is a related and relevant measure.

is placed anywhere in the area permitted by the reference set. For instance for the word *invites*, the permitted region can be defined as *invit^e+s* which means that the morph boundary + could alternatively be moved to before *e* —marked with ^— resulting in acceptance of either forms of *invite+s* and *invit+es*.The latter is a valid segmentation since it complies with other words based on the stem *invite* where the final "e" is removed from the stem before adding suffixes, e.g., *invit+ing*.

5.4 Motivation for a New Method

The BPRA approach has been used in several studies [VTS+11]; however, it has several important shortcomings. In most languages that have concatenative (or agglutinative) morphological processes, *it is not always clear* where the correct boundaries should be placed, because of the complexities in the morpho-phonology of the language.

For example, in Turkish, consider the morphology of *ekmeği*, composed of *ekmek* ('bread') + *i* (accusative marker); *k* changes to *ğ* due to regular consonant mutation. One *model*—i.e., one learning algorithm—might segment this word as *ekmeğ+i*, considering *ekmeğ* an allomorph of *ekmek* and *i* as the accusative marker, while another model might segment it as *ekme+ği*, considering *ekme* as an allomorph of *ekmek*, and *ği* an allomorph of *i*. Similarly, the plural in English: analyzing *flies*, one model could posit that the plural marker is *-s* (as in *dog+s*) and the stem *fly* has an allomorph *flie-*. Another model might posit an allomorph for the stem *fli-* and an allomorph *-es* for the plural marker. Clearly, there is no way to insist that one of the models is "better" or more correct than the other. This makes specifying the gold-standard segmentation problematic.

One option is to enumerate all acceptable segmentations for a word and give credit to the model if the predicted segmentation matches any of the reference segmentations. Alternatively, *fuzzy* boundaries can be defined for words by marking an acceptable *region* where the boundary can be placed. If the model predicts a boundary anywhere within the region, it receives credit. Such an evaluation method is suggested in [CL04b, CLLV05] as part of the Hutmegs package, as an attempt to provide gold-standard segmentations for Finnish. The main problem with this approach is that it is too permissive: it ignores the question of consistency and allows the model to violate its own decisions. A good evaluation scheme should penalize the model for *inconsistent* behavior.

In our Turkish example, *ekmeği*: both of the models discussed are equally good, as long as they remain faithful to the decision they make *throughout the entire corpus*, i.e., the model that prefers *ekmeğ+i* should do so for all words where consonants mutate before vowels, such as *ekmeğ+e*, *eteğ+i*, *artığ+ı*, etc.; if the model violates this (its own) decision on some words, it should be penalized.

5.5 Gold Standard Annotation

We now introduce our method: a *dilemma* is a (language-specific) situation where more than one segmentation may be acceptable according to the gold-standard annotation. A particular decision that a model makes in case of a dilemma, is called a *theory* that the model supports. For example, in English, the words *dogs* and *flies* may have gold-standard segmentations:

$$\begin{array}{cccccccccccc} & & & & & & & & X & & X & \\ d & o & g & + & s & & f & l & i & . & e & . & s \end{array}$$

Segmentation of *dogs* is unproblematic, the placement of the morph boundary is clear, marked with $+$. For *flies*, we wish to mark two alternative segmentations as acceptable. We do this by naming the dilemma, X, and specifying in the (language-specific) configuration that it has two acceptable theories, with boundary before **or** after e (not both). We indicate this by:

a. placing dots before and after e, each dot labeled by X,

b. specifying (in the configuration) in the definition of X that acceptable theories for X are 10 or 01.

This means that the word must be segmented as either *fli+es* or *flie+s*, respectively; 1 indicates the presence of a boundary, and 0 indicates absence of it.

The key point then is that if the evaluation corpus contains many similar words—*flies, tries, cries, supplies*, etc.—and they are all annotated similarly, then the model must segment *all* of these words according to theory 10 or according to theory 01—*consistently*. If the model is not consistent, it will be penalized. However, the penalty must be applied in such a way as to give the model under evaluation maximal benefit of the doubt. For instance, if the model's output resembles either of the columns

$$\begin{array}{rcr} flie + s & & fli + es \\ trie + s & \text{or} & tri + es \\ crie + s & & cri + es \\ supplie + s & & suppli + es \end{array}$$

it will get full credit for all four words (according to theory 01 for the column on the left or theory 10 in case of the column on the right). However if the model outputs the following as its answer to the segmentation problem:

$$\begin{array}{r} fli + es \\ trie + s \\ crie + s \\ supplie + s \end{array}$$

although these segmentations are correct *individually*, the model, as a whole is behaving inconsistently and should not be given full credit. There are two possible scenarios here. One is to assume that for this dilemma (X) the correct theory is 01, meaning that the accepted segmentation pattern is "$\cdots ie + s$", in which case $\frac{3}{4}$ of the words will yield full credit. The other case is to consider the theory 10 as correct, "$\cdots i + es$", which will result in full credit for $\frac{1}{4}$ of the words. Since these two theories are both valid according to the configuration, at evaluation time, the evaluation algorithm must find which theory results in the highest score for the model. In this example the theory supported by the majority of the words should be chosen as the reference.[11]. This is what we mean by assuring *maximal benefit of the doubt*. By following this approach, the evaluation algorithm does not discriminate against any model, since it assures maximal possible score for every model. The theory that yields the maximal score for a dilemma is referred to as the "*decision*" that model makes, or the *theory* that it *supports*.

Many other kinds of dilemmas and theories can be defined. Each dilemma is marked with its own unique label in the gold standard annotation, along with the theories it admits.

In our approach to evaluation of segmentations, given a dilemma, the model receives credit for selecting one of the theories that the gold standard accepts, in a consistent way.

[11] Although this is true for this dilemma, we show in Section 5.6 that majority does not always give maximal benefit of the doubt.

It is crucial to note that these dilemmas are independent from each other, i.e., supporting a theory in one dilemma does not restrict or encourage a theory of another dilemma.

Given a gold-standard annotation, if we fix some theory for each dilemma, we can generate the set of unambiguous *correct* segmentations for these fixed theories. The evaluation algorithm then computes the boundary precision, recall, and accuracy (BPRA) to score the model's segmentation in the standard way.

The crucial feature of our method is that it identifies which theories are *preferred* by the system, given the segmentations that are being evaluated. A theory is preferred, if choosing it would yield the highest score for the system. Since all theories are equally valid, the evaluation algorithm must find the one that maximizes the overall score.

Computing this exhaustively would require listing all possible segmentation sets based on all possible theories for each dilemma. This would make the problem intractable when the number of dilemmas is large, since it will require going through $O(n^k)$ potential reference segmentations, where n is the number of dilemmas and k is the maximum number of theories for a dilemma.

The overall score that we aim to maximize for the system is accuracy, since maximizing precision or recall separately results in preferring under-segmentation or over-segmentation, respectively. One could maximize the F-score to balance recall and precision, but because of the complicated and non-linear relation between F-score and the number of segmentation points, it is not obvious how to perform the computations with reasonable time complexity. We show how maximizing accuracy, under the assumption of independence of the theories, allows us to produce a consistent evaluation score.

5.5.1 Dilemmas and Labels

As mentioned above, our focus in this evaluation method is on dilemmas that a human expert/annotator encounters while annotating the segmentations—alternative ways to segment a word into morphs that are legitimate, and only depend on the annotator's decisions about allomorphy. These dilemmas can depend on what morphemes are present in the word. Each dilemma is identified by a unique label. There are labels of different *arities*, which will be explained briefly using examples.

5.5.2 Two-Way Dilemmas

This is the simplest type of possible dilemma. It is used when a human expert believes that placing a morph boundary between two symbols is optional, meaning that it is not absolutely necessary to segment it at that point. One example would be in cases where for historical reasons one might place a morph boundary between two potential morphs. In other words, the morph pair is old enough ("ossified") to be legitimately considered as a single morph. One example in Turkish is *çıkmaz* (meaning 'stalemate', 'dead end') which is considered by many as a single morph, but could be segmented as *çık* (from *çıkmak*; 'to exit') and suffix *-maz* (a form of negation in Turkish).

Another example is the Finnish word *"tulo"* (meaning 'coming, arrival')[12]

```
Y
tul.o
```

This means that the system under evaluation can decide either to segment at the points labeled `Y` or not to segment. Segmenting (placing a morph boundary) at this point corresponds to theory 1 and no boundary corresponds to theory 0; the valid theories for dilemma `Y` are 0 and 1. The practicality of this notation will become clear later, with labels of higher arity. This is declared in the gold-standard annotation, in the configuration file:

```
(Y 2 0 1)
```

The first element of the list is the symbol (label) that identifies the dilemma; the next element in the list is the arity (2); the remainder of the list enumerates the valid theories. `Y` is a dilemma with two possible theories/choices, of which (either!) `0` or `1` is an acceptable theory.

5.5.3 Four-Way and Higher-Arity Dilemmas

Sometimes in gold standard annotation we face dilemmas in which more than one potential boundary is involved; for example, *flies*, as above, will be annotated:

[12] In this example, it is not quite clear how to deal with the "*o*". Although in Finnish there exists a derivational suffix "*-o/-ö*" which is used to form nouns out of verbs—e.g. *tulla* (to come) → *tulo* (coming, arrival), here we could either have a fixed form *"tulo"*, or a form that is derived from the verb *"tulla"*.

```
X X
fli.e.s
```

It is important to note that since label `X` will be used in this type of dilemmas, it will *always* appear in pairs. Only one theory (ideally) should be selected as a segmentation decision by the system under evaluation, and for example if it chooses to segment at both points or neither, it should be penalized. To impose this kind of restriction, we use the following line in the label definition file:

```
(X 4 1 2)
```

the number 4 states that the dilemma `X` is a "4-way" dilemma, meaning that in total there are four possible theories—ways to perform the segmentation—that the system could follow; thus, there are $\log_2 4 = 2$ consecutive occurrences of `X` expected in the gold-standard words that contain this dilemma. The following numbers indicate which of the 4 possible segmentation theories are theoretically valid and can be chosen by the system without penalty. The numbers are decimal values of binary representations of the possible segmentation configurations, designating the presence of a morph boundary by 1 and lack of boundaries by 0. This is depicted in Table 2.

possible segmentation	**comparison to gold standard**	**numeric representation**
flie+s	`fli.e.s` `fli e+s`	$(01)_2 = 1$
fli+es	`fli.e.s` `fli+e s`	$(10)_2 = 2$

Table 2: Valid segmentation configurations for label `X` and word `fli.e.s`, with decimal representation. Segmenting (placing a morph boundary) at a point corresponds to `1` and not segmenting corresponds to `0`.

Of course there can be such dilemmas where the annotator decides it is also acceptable to segment at *both* locations, or neither. In such a case, corresponding numbers $0 = (00)_2$ and $3 = (11)_2$ can be added to the list of allowable theories.

Finally, this notation can be extended to dilemmas of higher order. For example the following configuration line would define a label for an 8-way dilemma (corresponding to 3 possible segmentation locations, hence, 3 consecutive labels in gold-standard

annotation):

```
(C 8 3 5 6)
```

The list (3 5 6) indicates that the *valid theories* are those which have exactly two (out of the three) segmentation points selected—corresponding to binary values (`011`, `101`, `110`).

The mark-up notation described here requires native-speaker skill as well as understanding of computational morphology, which makes it a complicated task that may take many man-hours to create. Nevertheless, it makes the evaluation task more objective and enables us to consider the ambiguities in evaluating morphological segmentations and favoring consistent models.

5.6 Performance Measure of Segmentations

Given a set of our gold standard segmentations, computing the BPRA measures is straightforward. Using the annotation method described above, we can choose an arbitrary theory for each dilemma, and generate the fixed gold-standard segmentations based on this set of theories. We want to give the model under evaluation the freedom to choose any of the valid theories, but force it to be consistent throughout the corpus and make the same decisions in similar situations. The model will be penalized for breaking its own decision.

As discussed above, we need a way to determine, for each dilemma, which theory is *preferred* by the model. In the "ideal" situation, where the model chooses one theory for each dilemma and segments all examples according to the chosen theory, it is clear that the preferred theory is the one that the model is complying with, and there is no penalization.

We next consider the case where for a given dilemma, D, the model decides to segment some instances according to one of the theories, d_1, but other instances according to another theory, d_2. Either of these theories could be chosen as the correct decision and the BPRA measures could be computed for them, but our goal is to give the model *maximum* benefit of the doubt. The theory that has been chosen most often throughout the data-set might seem to be the theory that should be preferred. Although this is true for one-way dilemmas, it is not always the case. The next example clarifies this:

The annotated gold standard for label `(Z 4 0 1 2 3)` along with an example of a model's response is shown in Figure 9. As easily seen, the first three cases are

Gold standard	Model response
Z Z 1. abc.d.e	abc d e
Z Z 2. fgh.i.j	fgh i j
Z Z 3. klm.n.o	klm n o
Z Z 4. pqr.s.t	pqr s+t
Z Z 5. uvw.x.y	uvw x+y
Z Z 6. zab.c.d	zab+c+d
Z Z 7. efg.h.i	efg+h+i

Figure 9: Left column: minimal example gold-standard annotation, with 7 instances of the 4-way dilemma Z. Right column: actual segmentations produced by a model to be evaluated using the gold standard; segmentation points are shown with +; blanks mark potential relevant segmentation points that were *not* segmented by the model (shown as blanks to help visualization).

segmented according to theory 00, the next two according to theory 01, and the last two according to theory 11; no word is segmented according to theory 10. It might seem that theory 00 is preferred by this particular model, but a closer look will falsify this. Let us compute the scores for each theory, assuming it is the preferred one. These score are shown in Table 3.

There are 7 words of length 5, so each has 4 potential segmentation boundaries. In total, there are 28 possible boundaries; 14 of these boundaries, which are not related to dilemma Z, are correctly left unsegmented by the model, regardless of the chosen theory. Depending on the preferred theory, some of the other boundaries are correct

or incorrect. For instance if theory `00` is chosen, for each of the first three words, both boundaries relevant to `Z` will be counted as correct, so there will be 3×2 additional correct boundaries. For the next two words, only one of the relevant boundaries (the first one) will be counted as correct, which will account for 2 (compare this to the number of *bits in common* between the preferred theory and the one supported by words 4,5). The last two words have been segmented incorrectly according to theory `00` of `Z`, so they will not contribute anything to the final accuracy score. Thus, the accuracy in case of preferring `00` will be $\frac{14+3\times2+2\times1+2\times0}{28} = \frac{22}{28} = 0.786$. The same process is followed for the second theory resulting in an accuracy score of 0.821. Although `00` has more supporters than `01`, it results in a lower accuracy.

The reason is that for 2-way and higher-order dilemmas, when one theory is preferred, words that have been segmented according to other theories—i.e., which support other theories—may still contribute to the accuracy score. This is because the theories are not disjoint, but have segmentation points in common and choosing one theory, might partially help other theories as well. Due to these indirect contributions to the total score, sometimes a theory with fewer supporters can surpass another theory with more supporters.

Theory	# Supporters	P	R	F	A
`00`	**3**	0	1	0	0.786
`01`	2	0.67	0.57	**0.62**	**0.821**
`10`	0	0.33	0.29	0.31	0.679
`11`	2	1.00	0.43	0.60	0.714

Table 3: Evaluation measures for the minimal example.

Thus, to give the model maximum benefit of the doubt, we need to choose the theory that maximizes one of the evaluation measures, not the number of supporters. Precision and recall are not good choices for maximization, since maximizing one of them will trade off with the other, resulting in under-segmentation or over-segmentation. F-score and accuracy are the remaining options. In our method, we give the models maximal benefit of the doubt in terms of accuracy since it is computationally easier, as shown below.

For this purpose, we need a score for dilemmas which maximizes accuracy. Let us denote the *set* of dilemmas as $\mathcal{D}$ and a valuation—i.e., an assignment of theories t to dilemmas in $\mathcal{D}$—as $[D]_t$ and the accuracy score under valuation t with A_t. When computing accuracy, the denominator $tp + tn + fp + fn$ is the number of possible

boundaries in the gold-standard data, which is constant and independent of the chosen theories. It is exactly equal to $\sum_{w \in W_{gs}} (|w| - 1)$ where W_{gs} is the set of gold standard words. Thus, we maximize the numerator and find:

$$\begin{aligned} \arg\max_{[[D]]_t} A_t &= \arg\max_{[[D]]_t} \frac{tp + tn}{tp + tn + fp + fn} \\ &= \arg\max_{[[D]]_t} (tp + tn) \end{aligned}$$

Since the ambiguities in words do not overlap, and each boundary in the gold-standard words can be marked with a maximum of one label, Dilemmas are independent of each other, and selecting one does not influence the others. Thus, we can maximize the sum of true positives and true negatives *separately for each dilemma*, and solve the problem by finding the preferred theory for one dilemma at a time.

	Contribution to $tp + tn$				
		`00`	`01`	`10`	`11`
Candidate theory	`00`	2	1	1	0
	`01`	1	2	0	1
	`10`	1	0	2	1
	`11`	0	1	1	2

Table 4: Contribution of each theory to $tp + tn$ score for all candidate theories, for every instance of the dilemma in the data. Each cell shows how many correct segmentation boundaries will be contributed to the accuracy score if the theory corresponding to the row is preferred, and the theory corresponding to the column is supported by the word.

To do this, for each dilemma, we create a "contribution" table similar to Table 4. Each row corresponds to one candidate theory; given that the theory corresponding to the row is the preferred one, the table shows how many correct boundaries would be contributed by supporters of theories in the columns. Since the dilemma in the Table is a 4-way dilemma, for each instance two boundaries should be examined. For any instance, if the preferred theory is the same as the supported theory, there are two correct boundaries. If a different theory than the preferred one is selected, depending on how many segmentation decisions they have *in common*, it will contribute 1 point or none to the numerator. The number of common segmentation decisions is given by the number of bits the binary representations of the two theories have in common which can be computed using the XNOR boolean operation.

If the number of supporters of each theory is counted (denoted by n_t for theory t), the result can be tabulated similarly to Table 5. The sum in each row shows what the contribution to the numerator will be if the corresponding candidate theory the preferred one. The task is now straightforward: select the theory for which the sum is maximized.

		Contribution to $tp + tn$			
		00	01	10	11
Candidate theory	00	$2n_0 +$	$n_1 +$	$n_2 +$	0
	01	$n_0 +$	$2n_1 +$	0 +	n_3
	10	$n_0 +$	0 +	$2n_2 +$	n_3
	11	0 +	$n_1 +$	$n_2 +$	$2n_3$

Table 5: Contribution of theories to $tp+tn$ for each candidate for the all occurrences of the dilemma. Each cell shows how many correct boundaries will be contributed to the accuracy score if the theory corresponding to the row is preferred by the whole data set and the theory corresponding to the column is supported by the word. n_t is the number of supporters of theory t in the data set. Binary values representing theories are displayed for simplicity.

Another potential measure to maximize benefit of the doubt is the F-score. However, it is more complex than the accuracy, since the denominator is not constant with respect to the chosen theories for dilemmas, thus it is not clear how one should find the preferred theory sets for each dilemma at a time without enumerating all possible valuations for dilemmas. In other words, accuracy can be written down as a sum of independent non-negative numbers, each relevant to one dilemma, therefore optimizing each and every number will result in optimization of accuracy. However, F-score cannot be decomposed into independent components in the same way.

One potential problem with maximizing accuracy is the so-called *accuracy paradox*, which arises when the distribution of the true classes is very unbalanced. For example when the set of true positives is smaller than the set of false positives, and the model decides to unconditionally predict negative (i.e., leave all words unsegmented). The same also can happen when the true negative set is smaller than the true positive set and the model decides to segment at every potential segmentation point. In these two examples, the model can achieve a higher accuracy, while the F-score will be lower. The same principle of maximal benefit of the doubt can be applied to cases where it is important to avoid this paradox, however the optimiza-

tion technique presented here for accuracy will not be suitable to apply directly to F-score. We choose accuracy for optimization in this thesis, and will explore optimization of F-score as part of our future work.

5.7 Evaluation Algorithm

Given a set of segmented words (output of a model) and a gold standard annotation, as explained earlier, we first determine what decisions the model has made for each dilemma. This is done via an alignment algorithm which aligns symbols of the gold standard segmentation and the system response to find if there are any morph boundaries corresponding to the "." symbols in the gold standard. An example of such an alignment is shown in Figure 10.

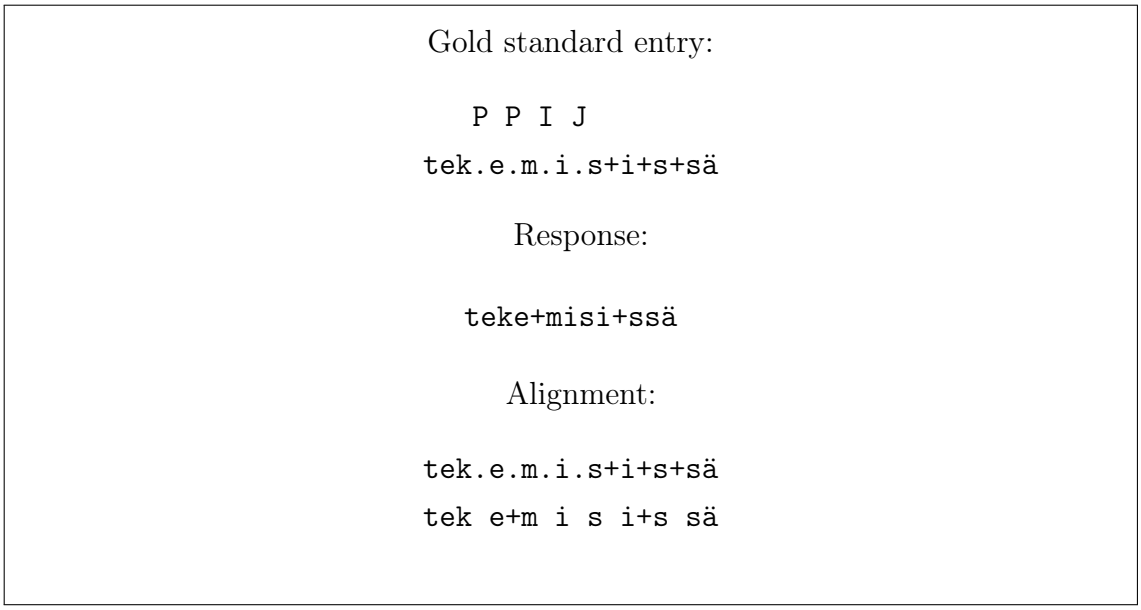

Figure 10: An example of a gold standard to response alignment. Alignment of "`.e.`" to "` e+`" shows that this particular word prefers theory 01 for dilemma P. Similarly, alignment of "`.`" to "` `" shows a preference of theory 0 for dilemmas I and J.

The overall schema of the evaluation algorithm can be summarized as follows:

1. For each dilemma D:
   (a) List all the words from the segmentation set that are relevant to D.

   (b) For each such word, find the supported theory by aligning the word to the gold standard entry.

   (c) Count the number of supporters of each theory.

   (d) Determine which choice of the valid theories yields the maximum accuracy, similarly to Table 5.

2. Once decisions for all dilemmas are made, generate the final reference segmentations (one segmentation per word) based on the decisions.

3. Use the standard method to calculate precision, recall, F-score and accuracy.

5.8 Empirical Test of the Evaluation Algorithm

Experiments with evaluating consistent and inconsistent models using our evaluation method demonstrate the two key features:

1. If we have an *inconsistent* model, standard BPRA will give it a *higher* score than our method.
2. If we have two competing models, our method will give a higher score to the model that is *more consistent*.

For the experiments that we designed to check the validity and effectiveness of our evaluation methodology, we generated two segmentation sets from a gold standard annotated for a Finnish corpus, consisting of about 1000 words. Both sets contain valid segmentations for each word viewed *separately*, however one of the sets is inconsistent in choosing the alternative segmentation points. The consistent set is generated by fixing one of the valid theories for each dilemma, and segmenting the words according to the chosen theory. For generating the inconsistent set, for each word and for each dilemma relevant to that word, one of the valid theories is chosen at random, and the word is segmented according to the decisions. Other segmentation boundaries are left untouched.

By definition, both segmentation sets will get the full score using the standard BPRA method, since all plausible segmentations are listed as alternatives in the gold standard, and any segmentation that is in that list will be accepted.

The evaluation results for the two segmentation sets with our new method are shown in Table 6. In addition to precision, recall, F-score and accuracy, several other statistics which are used to calculate the scores are also included. As can be seen,

the consistent model gets full credit in all measures, whereas the inconsistent model is penalized.

	Consistent	**Inconsistent**
$\|G\|$	2617	2635
$\|S\|$	2617	2600
$\|G \cap S\|$	2617	2471
$tp + tn$	8212	7919
$tp + tn + fp + fn$	8212	8212
Precision	100%	95.04%
Recall	100%	93.78%
F-Score	100%	94.40%
Accuracy	100%	96.43%

Table 6: Evaluation results: consistent and inconsistent segmentation sets. $|G|$ is the number of segmentation points in the final reference set (based on gold standard and model's decisions), $|S|$ is the number of segmentation points in the model's response, $tp+tn$ denotes the number of matching boundaries, and $tp+tn+fp+fn$ is the number of potential segmentation boundaries.

6 Experiments and Results

In this chapter several variations of the algorithm are introduced, followed by results of the algorithm on Finnish, Turkish, and Russian data-sets. The introduced variations are motivated by linguistic universal principles which are believed to apply to all or most languages. Our results are also compared to one of the state-of-the-art methods, Morfessor CatMAP [CL05]. The evaluations are performed as explained in the previous chapter.

A visualization of the model output trained with Finnish data with 5 states along with sample segmentations and an extract of the output lexicon can be found in Appendix 1.

6.1 Directional Model

The states in our algorithm are ultimately meant to reflect morpheme classes, such that in the ideal case each state will emit morphs that are categorically the same;

for instance noun stems may be emitted from state X, verb stems from state Y, nominal case endings for a given type of nominal stems from state Z, etc. The morphotactics of any language specify exactly the order in which morphological classes may follow one another. For example in English, the plural morpheme for nouns—"*s*" or "*es*"—appears always at the end of the word. In Russian, e.g., a word w can have one or more prefix, then a stem, then one or more suffixes—always in a fixed order. Further, different kinds of suffixes have strict ordering among them—e.g., derivation precedes inflection. Thus, the order of classes in the lexicon is fixed. This is without taking into account *compounding*.

This property of languages can be reflected in our model by restricting the FSM to be directional, i.e., from each state the model is allowed to transit only to later states. The order is encoded in the labels of the states, meaning that from state S_k the model can jump to state S_j only if $j > k$. To enforce directionality in the model, in Equation (14) the range of l changes to $0 \leq l < j$ and we have instead

$$C(\sigma^i, S_j) = \min_{\substack{0<k<i \\ 0\leq l<j}} \Big\{ C(\sigma^k, S_l) + C_{Trans}(S_l, S_j) + C_{Emit}(\sigma^i_{k+1}|S_j) \Big\}. \tag{22}$$

To reflect this in the dynamic programming, we constrain the dynamic programming matrix so that the preceding state l in line 18 of the algorithm on page 30 ranges from 0 only up to $j - 1$, rather than up to K. Since the states are ordered, we cannot transit from a later state to an earlier one. The idea is that since this reflects the nature of the languages, this way we are somehow *guiding* the search procedure towards areas in the search space that are more likely to have the optimum we look for or at least to avoid the subspace that is less likely to contain good solutions.

As a result of these changes, the event space for prequential coding of transitions is also limited to transitions to later states only.

6.2 Natural Classes of Morphs

Another general principle is that morph can be classified into two principal kinds: *stems* vs. *affixes*. We fix some range of states in the beginning to be *prefix* states, followed by a range of *stem* states, followed by *suffix* states. The heuristic is then that for no word the model can directly transit from a state associated with prefixes to a suffix state, and the dynamic programming path has to pass through at least one stem state reflecting the fact each word must have at least one stem. This,

of course, is added on top of *directionality* of the model discussed above. In our experiments, e.g., we divide the 15 available states into 2 classes for prefixes, 6 for stems, and 7 for suffixes. Similar to the directional model, the event space of the transitions changes accordingly.

6.3 Bulk Resegmentation

Another universal linguistic principle we use is that stems and affixes have very different properties. Stems classes are *open*—i.e., potentially very large, while affix classes are necessarily *closed*—very limited. This is reflected, e.g., in borrowing: a language may borrow any number of stems from other languages freely, whereas it is extremely unlikely to borrow a suffix.

Conversely, in general a randomly chosen affix is typically expected to occur *much* more frequently in the corpus than a random stem. This is true in general, although a language may have some exceptionally rarely used affix, which might happen to be less frequent than a very frequent stem.

Based on this principle, we introduce another heuristic to guide the search: after normal resegmentation, all states are checked for "bad" morphs that violate this principle—very frequent morphs in stem classes and very rare morphs in affix classes. This introduces two hyper-parameters into the model: s_{max} for maximum tolerated count of a stem, and a_{min} for minimum frequency of an affix (we set both to 100). With a certain probability $\pi(T)$ for each potential bad morph, all words that contain that morph are removed from the model *in bulk* (from the lexicon, and their transition and emission counts), and resegmented afresh. Probability $\pi(T)$ depends only on the simulated annealing temperature T, so that when T is high, $\pi(T)$ is small; as T cools, $\pi(T) \to 1$, hence always removing bad morphs and resegmenting their corresponding words when the algorithm is completely greedy. Several different schedules have been tried for $\pi(T)$ such as exponentially increasing, logarithmically increasing, and linearly increasing functions. The motivation behind this heuristic is to reduce influence of *bad* morphs on the words in the resegmentation process and give the search procedure the ability to escape such traps with the hope of avoiding local optima.

6.4 Results

We have tried the algorithm on 3 languages: Finnish, Turkish, and Russian, each from a different language family—respectively Finno-Ugric, Turkic, and Indo-European. For each language several sources that are publicly available have been processed to extract list of words. These sources are mostly books such as novels. We have avoided using publicly available news or media text since that kind of text tends to be homogeneous which might not reflect all properties of the language. We have extracted several data-sets which are summarized in Table 7.

Name	**Language**	**Number of words**
`fi-80k`	Finnish	82 808
`tr-90k`	Turkish	90 719
`ru-100k`	Russian	104 731

Table 7: Extracted data-sets for Finnish, Turkish, and Russian

The results of the experiments are presented in Figures 11, 12, and 13, for Finnish, Russian, and Turkish. Each point in the plots represents a single run of the algorithm. The coordinates of each point are its recall and precision, and the accuracy for each point is in its label. The Morfessor CatMAP algorithm [CL05] was run on the same data-sets for comparison. We use this version of Morfessor, since it obtained the best performance over all Morfessor variants, as stated in [GVSK14].

Morfessor's CatMAP algorithm has a parameter, b, called the "perplexity threshold, which indicates the point where a morph is as likely to be a prefix as a non-prefix" [CL05]. This parameter trades off recall and precision; the more data there is, the higher b should be; the higher b is, the less words are split, leading to higher precision but lower recall. We ran Morfessor with b varying from 5 to 800, which yields the red (solid) line in the plots.

Currently, our model has several hyper-parameters: The probability ρ of placing a morph boundary between any two adjacent symbols during the initial random segmentation (currently 0.20–0.25); the number of classes K (currently 15); the assignment of classes to prefix, stem and suffix kinds: s_{max} and a_{min}. For simulated annealing, the cooling schedule α (currently 0.995), initial and final temperature, T_0 and T_F, were roughly tuned so that the cost curve across iterations has a characteristic sigmoid shape, as it was shown in Figure 6 (as commonly done in simulated annealing). The blue points (stars) in the plots correspond to runs of our method,

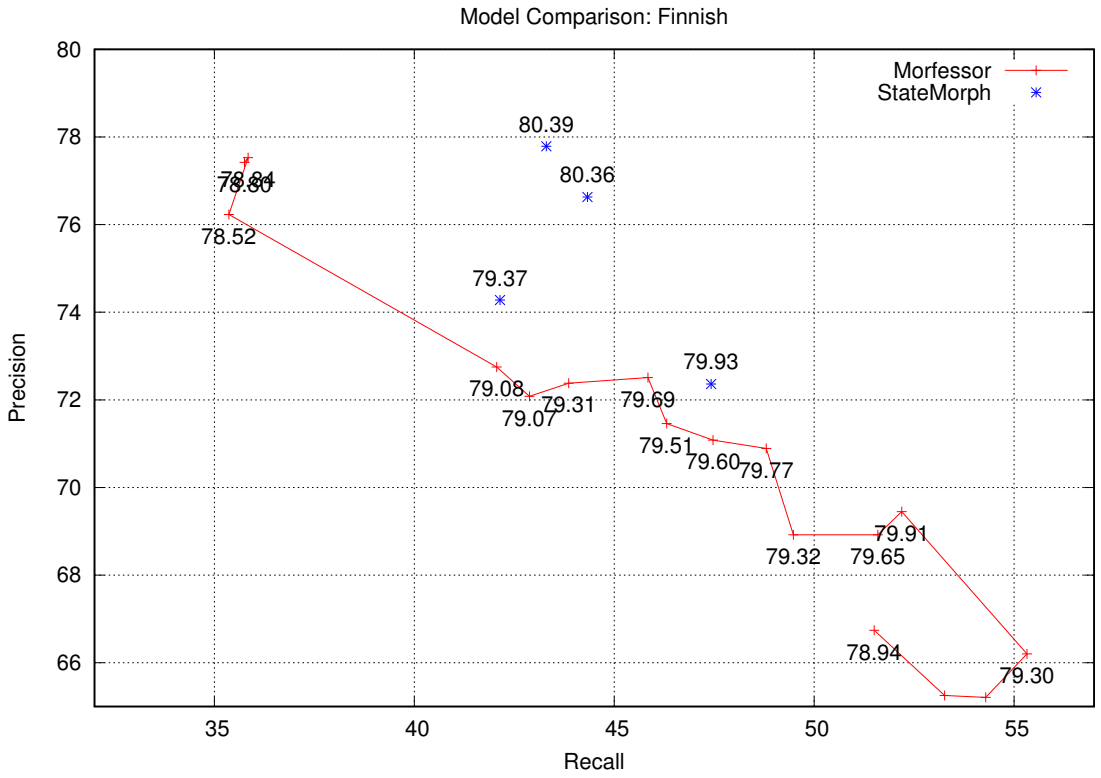

Figure 11: Precision vs. recall for Finnish data

with different settings of the hyper-parameters. Since our method does not have any parameter to directly balance precision and recall, there is no order between them and they are just represented as single points. These parameters have not been tuned jointly; rather we started with certain "sensible" settings for the parameters (above) and checked the effect of varying the parameters independently. They can be further optimized (e.g., on a development corpus), which can be investigated in detail in future work. However, even as they stand now, all runs of StateMorph show a substantial improvement in terms of recall and precision over the best Morfessor model: the blue points always lie above the red curve in the plots. That means that, e.g., at a given level of recall, StateMorph always has higher precision. For Finnish, the gain in precision is 2–8%; for Russian, it is 15% on average; for Turkish, 2–7%.

Conversely, at a given level of precision, StateMorph always has higher recall, hence higher F-score; for very large b, Morfessor reaches higher recall, but at a substantial loss in precision. The accuracy of StateMorph is also mostly better than Morfessor's, though not always. At a given level of recall, StateMorph always achieves higher accuracy. The fine-grained interaction between the the hyper-parameters and

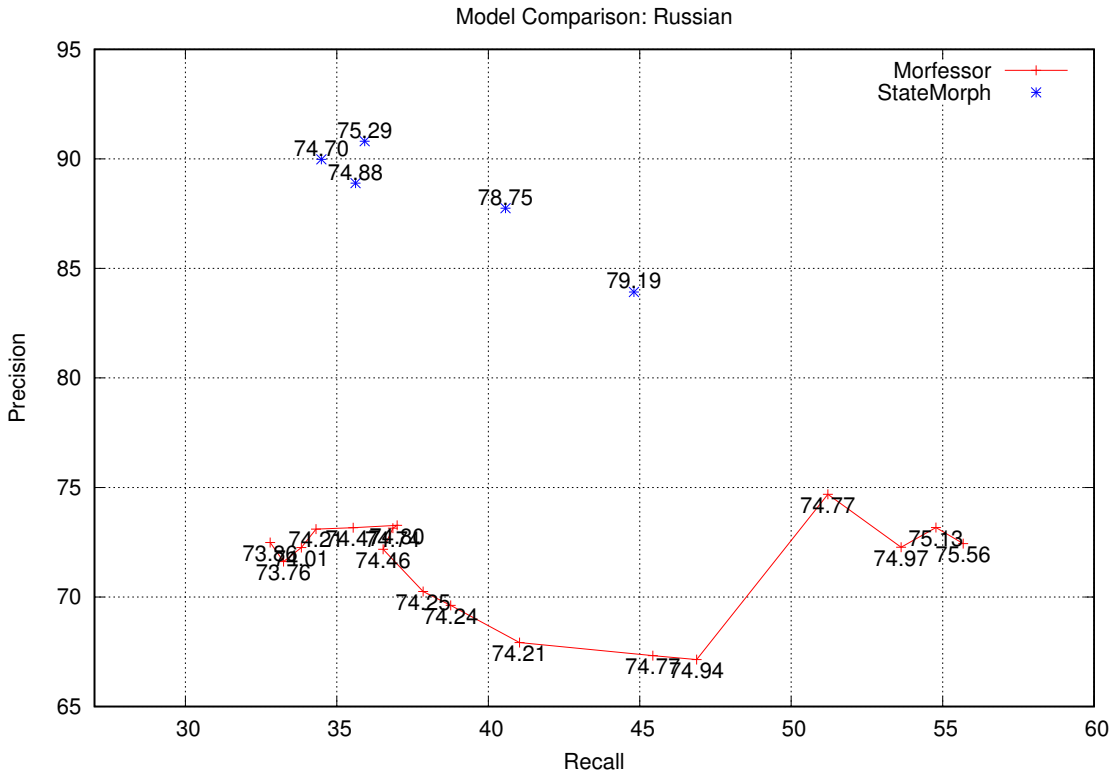

Figure 12: Precision vs. recall for Russian data

performance remains to be explored.

Qualitative evaluation of classification: A crucial point about StateMorph is that it learns classes of very high quality. A sample of an example run of StateMorph can be found in Appendix 1. Manual inspection of the output shows that each state groups together morphs of highly similar natures. As can be seen in Appendix 1, several large states emit morphs that are generally verb or noun stems. Some other states emit suffixes that are of similar kinds. These states have smaller number of distinct morphs with higher frequencies compared to the stem states.

As is natural for MDL, when several affixes appear frequently together, they may be learned as a single affix; this explains lower recall. However, this problem may be addressable as a post-processing step, after the learning is complete (in future work). Of course, evaluating classes quantitatively is difficult, hence we evaluate quantitatively the segmentations only.

Running time comparison: The running time of StateMorph depends on the data size and the chosen simulated annealing cooling schedule. The runs reported here take 8 to 10 hours to finish using the parallelized version with 40 machines.

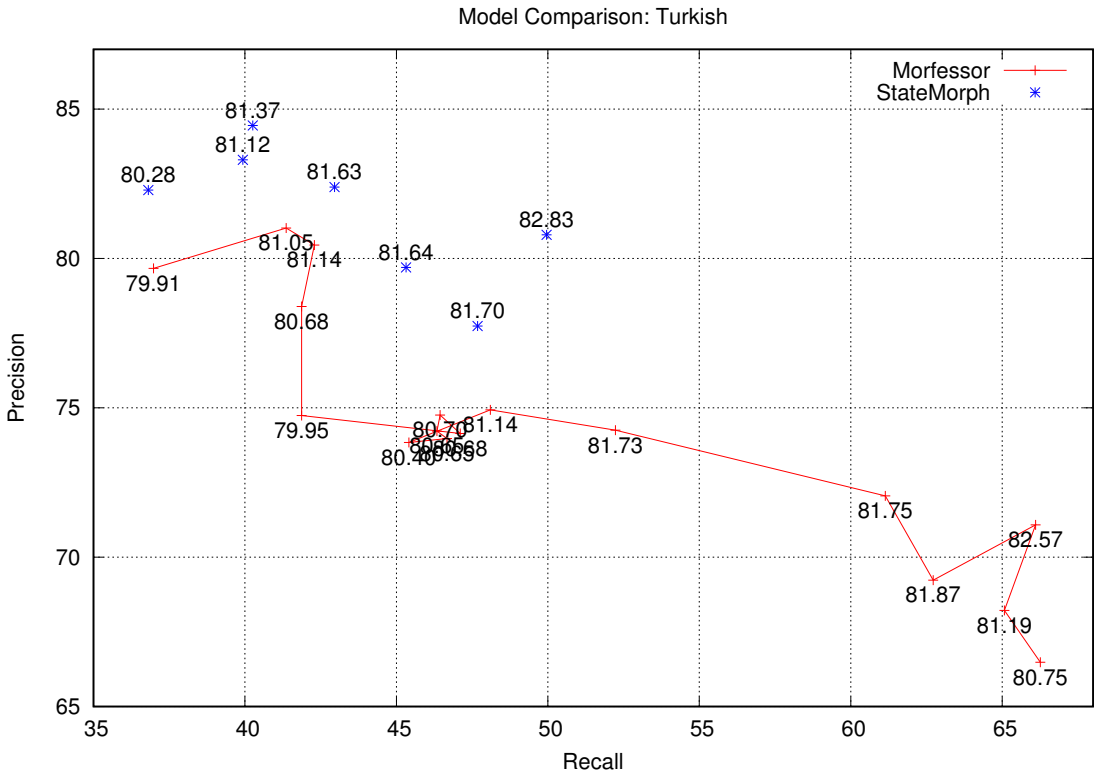

Figure 13: Precision vs. recall for Turkish data

Parallelization does not divide total running time by the number of machines, since there is considerable overhead due to synchronization at each iteration and instability of the network on the cluster. However, parallelization was a necessary step since the non-parallelized versions of StateMorph had a very long running time even with faster simulated annealing cooling schedules. In our experiments, each Morfessor run on our data sets took less than an hour to finish.

7 Conclusions and Future Work

We have presented an algorithm for automatic segmentation of a corpus of words. The proposed model tries to approach the problem in a systematic way, by grouping the discovered morphs into classes that respect several general linguistic principles. Using no prior knowledge about the language, starting from a randomly initialized model, the algorithm learns the segmentation from the data by optimizing a cost function following the MDL principle. The results obtained with the linguistically motivated heuristics consistently outperform the current state of the art.

In future work, we envision several improvements. For the cost function, a natural improvement is coding the lexicon more efficiently, by taking into account symbol frequencies. We plan to explore learning the number of classes automatically, which should be reflected in the code-length and the algorithm should ultimately find the most suitable number of classes that could be associated with linguistically defined morphological classes for each language.

Another contribution of this work is a novel method and corresponding resources for evaluation of morphological segmentations. The features discussed in Chapter 5 distinguish our proposed evaluation method from methods used in prior work—where a model could not be required to segment data in a consistent fashion. We note that "ambiguity" is inherent in morphological segmentation: it is impossible to posit a single segmentation that is "correct" in many cases in many languages. Thus the gold standard must provide *flexibility* to accommodate different theories of morphology. However, if two models—where one is consistent and one is inconsistent—receive equal score, that means that the evaluation method being used is not informative.

Our evaluation method, while it resolves the matter of consistency, still leaves certain problematic cases unresolved. We briefly discuss them here.

One case is what we consider a true *morphological ambiguity*: a word that can be legitimately segmented in more than one way. For example, Turkish *evini* (*ev*: 'home/house') can be analyzed as *ev*(noun stem)+(2nd person singular possessive suffix)+ACC (accusative marker) or *ev*(noun)+(2nd person singular possessive suffix)+ACC (accusative marker). Ambiguity arises because *+in* is the 2nd person singular possessive marker and *+i* is 3rd person singular possessive marker, but the morphology requires an epenthetic *+n* after a 3rd person singular possessive suffix as a "buffer" consonant before certain suffixes are added—in this case, the accusative marker, which begins with a vowel. This applies to all similar word endings, such as *+ını*, *+unu*, and *+ünü*; which one appears in an instance is determined by the rules of Turkish vowel harmony; this also occurs with other suffixes, such as dative, ablative, and other nominal cases.

We distinguish regular ambiguity from sporadic ambiguity. An ambiguity is regular if, as in the example above, there are many words that follow the same pattern, and have the same ambiguity. We plan to extend the current algorithm to address true ambiguities in future work.

Acknowledgments

My sincere thanks goes to my supervisor Dr. Roman Yangarber for the continuous support and guidance in the course of this thesis work as well as to Dr. Jyrki Kivinen for the thorough comments and suggestions. I would like to express my deepest gratitude to Barış Serim and Suzan Bayhan for help with Turkish gold standard annotations, as well as to Kirill Reshetnikov for annotation of Finnish and Russian evaluation set. I would like to further thank Teemu Roos for the useful discussions and precious suggestions. I would like to acknowledge contributions of Suvi Hiltunen who had worked on the earlier versions of the algorithm. I would like to recognize the financial support received from UraLink and FinUgRevita projects of the Academy of Finland, which made this work possible. Finally, my greatest gratitude goes to my dear family and friends for their continuous and endless love and support.

References

AAAK04 Argamon, S., Akiva, N., Amir, A. and Kapah, O., Efficient Unsupervised Recursive Word Segmentation Using Minimum Description Length. *Proceedings of the 20th International Conference on Computational Linguistics*, COLING '04, Stroudsburg, PA, USA, 2004, Association for Computational Linguistics, pages 1058–1064.

AF11 Aronoff, M. and Fudeman, K., *What is morphology*, volume 8. John Wiley & Sons, 2011.

Ber06 Bernhard, D., Unsupervised Morphological Segmentation Based on Segment Predictability and Word Segments Alignment. *In Proceedings of the Pascal Challenges Workshop on the Unsupervised Segmentation of Words into Morphemes*, 2006.

Ber08 Bernhard, D., Simple Morpheme Labelling in Unsupervised Morpheme Analysis. In *Advances in Multilingual and Multimodal Information Retrieval*, Peters, C., Jijkoun, V., Mandl, T., Müller, H., Oard, D. W., Peñas, A., Petras, V. and Santos, D., editors, volume 5152 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2008, pages 873–880.

BMT02 Baroni, M., Matiasek, J. and Trost, H., Unsupervised Discovery of Morphologically Related Words Based on Orthographic and Semantic Similarity. *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning - Volume 6*, MPL '02, Stroudsburg, PA, USA, 2002, Association for Computational Linguistics, pages 48–57.

Çöl10 Çöltekin, Ç., A freely available morphological analyzer for Turkish. *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC 2010)*, 2010, pages 820–827.

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

CL02 Creutz, M. and Lagus, K., Unsupervised discovery of morphemes. *Proceedings of the ACL-02 workshop on Morphological and phonological learning-Volume 6*. Association for Computational Linguistics, 2002, pages 21–30.

CL04a Creutz, M. and Lagus, K., Induction of a simple morphology for highly-inflecting languages. *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*. Association for Computational Linguistics, 2004, pages 43–51.

CL04b Creutz, M. and Lindén, K., Morpheme segmentation gold standards for Finnish and English. Technical Report A77, Helsinki University of Technology, 2004.

CL05 Creutz, M. and Lagus, K., Inducing the morphological lexicon of a natural language from unannotated text. *Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR'05)*, volume 1, 2005, pages 51–59.

CLLV05 Creutz, M., Lagus, K., Lindén, K. and Virpioja, S., Morfessor and Hutmegs: Unsupervised morpheme segmentation for highly-inflecting and compounding languages. *Proceedings of the Second Baltic Conference on Human Language Technologies*, Tallinn, Estonia, 2005, pages 356–370.

CM14 Can, B. and Manandhar, S., Methods and Algorithms for Unsupervised Learning of Morphology. In *Computational Linguistics and Intelligent Text Processing*, Gelbukh, A., editor, volume 8403 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2014, pages 177–205.

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

De 59 De La Briandais, R., File Searching Using Variable Length Keys. *Papers Presented at the the March 3-5, 1959, Western Joint Computer Conference*, IRE-AIEE-ACM '59 (Western), New York, NY, USA, 1959, ACM, pages 295–298.

Dem07 Demberg, V., A language-independent unsupervised model for morphological segmentation. *In Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 2007, pages 920–927.

DG08 Dean, J. and Ghemawat, S., MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51,1(2008), pages 107–113.

DLR77 Dempster, A. P., Laird, N. M. and Rubin, D. B., Maximum likelihood from incomplete data via the em algorithm. *journal of the Royal Statistical Society, Series B*, 39,1(1977), pages 1–38.

DN07 Dasgupta, S. and Ng, V., High-Performance, Language-Independent Morphological Segmentation. *NAACL HLT 2007: Proceedings of the Main Conference*, 2007, pages 155–163.

ELZ+10 Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J. and Fox, G., Twister: A Runtime for Iterative MapReduce. *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, New York, NY, USA, 2010, ACM, pages 810–818.

Gau99 Gaussier, É., Unsupervised learning of derivational morphology from inflectional lexicons. *Proceedings of the Workshop on Unsupervised Learning in Natural Language Processing*. Association for Computational Linguistics, 1999, pages 24–30.

Gol01 Goldsmith, J., Unsupervised Learning of the Morphology of a Natural Language. *Comput. Linguist.*, 27,2(2001), pages 153–198.

Grü07 Grünwald, P. D., *The minimum description length principle*. MIT press, 2007.

GVSK14 Grönroos, S.-A., Virpioja, S., Smit, P. and Kurimo, M., Morfessor FlatCat: An HMM-based method for unsupervised and semi-supervised learning of morphology. *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics*, Dublin, Ireland, 2014, Dublin City University and Association for Computational Linguistics, pages 1177–1185.

Har55 Harris, Z. S., From Phoneme to Morpheme. *Language*, 31,2(1955), pages pp. 190–222.

HB11 Hammarström, H. and Borin, L., Unsupervised Learning of Morphology. *Computational Linguistics*, 37,2(2011), pages 309–350.

Hil12 Hiltunen, S., Minimum description length modeling of etymological data. Master's thesis, University of Helsinki, 2012.

HS13 Haspelmath, M. and Sims, A., *Understanding morphology*. Routledge, 2013.

JM03 Johnson, H. and Martin, J., Unsupervised Learning of Morphology for English and Inuktitut. *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 2003–short Papers - Volume 2*, NAACL-Short '03, Stroudsburg, PA, USA, 2003, Association for Computational Linguistics, pages 43–45.

KCL06 Kurimo, M., Creutz, M. and Lagus, K., editors. *Proceedings of the PASCAL Challenge Workshop on unsupervised segmentation of words into morphemes*, Venice, Italy, 2006. PASCAL European Network of Excellence.

KCV+06 Kurimo, M., Creutz, M., Varjokallio, M., Arisoy, E. and Saraçlar, M., Unsupervised segmentation of words into morphemes–challenge 2005:

An introduction and evaluation report. *Proceedings of the PASCAL Challenge Workshop on Unsupervised segmentation of words into morphemes*, 2006.

KCV07 Kurimo, M., Creutz, M. and Varjokallio, M., Unsupervised Morpheme Analysis Evaluation by a Comparison to a Linguistic Gold Standard-Morpho Challenge 2007. *CLEF (Working Notes)*, 2007.

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

KK97 Karlsson, F. and Karttunen, L., Subsentencial Processing. In *Survey of the State of the Art in Human Language Technology*, Cole, R., editor, Cambridge University Press, New York, NY, USA, 1997, pages 96–100.

KM01 Kazakov, D. and Manandhar, S., Unsupervised Learning of Word Segmentation Rules with Genetic Algorithms and Inductive Logic Programming. *Machine Learning*, 43,1-2(2001), pages 121–162.

Kol63 Kolmogorov, A. N., On tables of random numbers. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 369–376.

KP06 Keshava, S. and Pitler, E., A simpler, intuitive approach to morpheme induction. *Proceedings of 2nd Pascal Challenges Workshop*, 2006, pages 31–35.

KTV08 Kurimo, M., Turunen, V. and Varjokallio, M., Overview of Morpho challenge 2008. In *Evaluating Systems for Multilingual and Multimodal Information Access*, Springer Berlin Heidelberg, 2008, pages 951–966.

KVT+09 Kurimo, M., Virpioja, S., Turunen, V. T., Blackwood, G. W. and Byrne, W., Overview and results of morpho challenge 2009. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, Springer, 2009, pages 578–597.

KVTL10 Kurimo, M., Virpioja, S., Turunen, V. and Lagus, K., Morpho Challenge 2005-2010: Evaluations and Results. *Proceedings of the 11th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology*, Uppsala, Sweden, jul 2010, Association for Computational Linguistics, pages 87–95.

Läm08 Lämmel, R., Google's MapReduce programming model – revisited. *Science of Computer Programming*, 70,1(2008), pages 1 – 30.

LFL98 Landauer, T. K., Foltz, P. W. and Laham, D., An introduction to latent semantic analysis. *Discourse processes*, 25,2-3(1998), pages 259–284.

LL10 Lavallée, J.-F. and Langlais, P., Unsupervised Morphological Analysis by Formal Analogy. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, Peters, C., Di Nunzio, G. M., Kurimo, M., Mandl, T., Mostefa, D., Peñas, A. and Roda, G., editors, volume 6241 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2010, pages 617–624.

Mat74 Matthews, P. H., *Morphology. An Introduction to the Theory of Word-structure*. Cambridge University Press, Cambridge, 1974.

MCLL09 Monson, C., Carbonell, J., Lavie, A. and Levin, L., ParaMor and Morpho Challenge 2008. In *Evaluating Systems for Multilingual and Multimodal Information Access*, Peters, C., Deselaers, T., Ferro, N., Gonzalo, J., Jones, G. J. F., Kurimo, M., Mandl, T., Peñas, A. and Petras, V., editors, volume 5706 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2009, pages 967–974.

MEB09 Moon, T., Erk, K. and Baldridge, J., Unsupervised Morphological Segmentation and Clustering with Document Boundaries. *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, EMNLP '09, Stroudsburg, PA, USA, 2009, Association for Computational Linguistics, pages 668–677.

MLCL04 Monson, C., Lavie, A., Carbonell, J. and Levin, L., Unsupervised Induction of Natural Language Morphology Inflection Classes. *Proceedings of the 7th Meeting of the ACL Special Interest Group in Computational Phonology: Current Themes in Computational Phonology and Morphology*, SIGMorPhon '04, Stroudsburg, PA, USA, 2004, Association for Computational Linguistics, pages 52–61.

MLCL08 Monson, C., Lavie, A., Carbonell, J. and Levin, L., Evaluating an Agglutinative Segmentation Model for ParaMor. *Proceedings of the Tenth Meeting of ACL Special Interest Group on Computational Morphology*

and Phonology, SigMorPhon '08, Stroudsburg, PA, USA, 2008, Association for Computational Linguistics, pages 49–58.

Mon04 Monson, C., A Framework for Unsupervised Natural Language Morphology Induction. *Proceedings of the ACL 2004 Workshop on Student Research*, ACLstudent '04, Stroudsburg, PA, USA, 2004, Association for Computational Linguistics.

MTF11 Mundkur, P., Tuulos, V. and Flatow, J., Disco: a computing platform for large-scale data analytics. *Proceedings of the 10th ACM SIGPLAN workshop on Erlang*. ACM, 2011, pages 84–89.

NF02 Neuvel, S. and Fulop, S. A., Unsupervised Learning of Morphology Without Morphemes. *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning - Volume 6*, MPL '02, Stroudsburg, PA, USA, 2002, Association for Computational Linguistics, pages 31–40.

NPY13 Nouri, J., Pivovarova, L. and Yangarber, R., MDL-based models for transliteration generation. *Proceedings of the First International Conference on Statistical Language and Speech Processing, SLSP 2013*, Dediu, A.-H., Martín-Vide, C., Mitkov, R. and Truthe, B., editors, Tarragona, Spain, 2013, pages 200–211.

NY14 Nouri, J. and Yangarber, R., Measuring language closeness by modeling regularity. *Proceedings of the EMNLP 2014 Workshop on Language Technology for Closely Related Languages and Language Variants*, Doha, Qatar, 2014.

NY16 Nouri, J. and Yangarber, R., A novel method for evaluation of morphological segmentation. *Proceedings of the 10th edition of the Language Resources and Evaluation Conference (LREC'16)*, Portorož, Slovenia, May 2016.

PCT09 Poon, H., Cherry, C. and Toutanova, K., Unsupervised morphological segmentation with log-linear models. *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, NAACL '09, Stroudsburg, PA, USA, 2009, Association for Computational Linguistics, pages 209–217.

PO09 Piskorski, J. and Others, Morphisto-an open source morphological analyzer for German. *Finite-state Methods and Natural Language Processing: Post-proceedings of the 7th International Workshop FSMNLP; Edited by Jakub Piskorski, Bruce Watson and Anssi Yli-Jyr{ä}*, volume 191. IOS Press, 2009, page 224.

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

SB08 Snyder, B. and Barzilay, R., Unsupervised multilingual learning for morphological segmentation. *ACL*, 2008, pages 737–745.

Sha01 Shannon, C. E., A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5,1(2001), pages 3–55.

SJ00 Schone, P. and Jurafsky, D., Knowledge-free Induction of Morphology Using Latent Semantic Analysis. *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*, ConLL '00, Stroudsburg, PA, USA, 2000, Association for Computational Linguistics, pages 67–72.

SJB02 Snover, M. G., Jarosz, G. E. and Brent, M. R., Unsupervised Learning of Morphology Using a Novel Directed Search Algorithm: Taking the First Step. *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning - Volume 6*, MPL '02, Stroudsburg, PA, USA, 2002, Association for Computational Linguistics, pages 11–20.

SKD02 Sharma, U., Kalita, J. and Das, R., Unsupervised Learning of Morphology for Building Lexicon for a Highly Inflectional Language. *Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning - Volume 6*, number July in MPL '02, Stroudsburg, PA, USA, 2002, Association for Computational Linguistics, pages 1–10.

SKD08 Sharma, U., Kalita, J. K. and Das, R. K., Acquisition of Morphology of an Indic Language from Text Corpus. *ACM Transactions on Asian Language Information Processing (TALIP)*, 7,3(2008), pages 9:1—-9:33.

SM10 Spiegler, S. and Monson, C., EMMA: A Novel Evaluation Metric for Morphological Analysis. *Proceedings of the 23rd International Conference on Computational Linguistics*, number August in COLING '10,

Stroudsburg, PA, USA, 2010, Association for Computational Linguistics, pages 1029–1037.

TMR10 Tchoukalov, T., Monson, C. and Roark, B., Morphological Analysis by Multiple Sequence Alignment. In *Multilingual Information Access Evaluation I. Text Retrieval Experiments*, Peters, C., Di Nunzio, G. M., Kurimo, M., Mandl, T., Mostefa, D., Peñas, A. and Roda, G., editors, volume 6241 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2010, pages 666–673.

VHB97 Voorhees, P., Hoffman, D. M. and Barnes, C., TREC information retrieval: Text research collection. vol. 5, 1997.

VTS+11 Virpioja, S., Turunen, V. T., Spiegler, S., Kohonen, O. and Kurimo, M., Empirical comparison of evaluation methods for unsupervised learning of morphology. *TAL*, 52,2(2011), pages 45–90.

Wet13 Wettig, H., *Probabilistic, Information-Theoretic Models for Etymological Alignment*. Ph.D. thesis, University of Helsinki, 2013.

Whi09 White, T., *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., first edition, 2009.

WNRY12 Wettig, H., Nouri, J., Reshetnikov, K. and Yangarber, R., Information-theoretic methods for analysis and inference in etymology. *Proceedings of WITMSE-2012: the Fifth Workshop on Information-theoretic Methods in Science and Engineering*, De Rooij, S., Kotłowski, W., Rissanen, J., Myllymäki, P., Roos, T. and Yamanishi, K., editors, Amsterdam, the Netherlands, 2012.

WNRY13 Wettig, H., Nouri, J., Reshetnikov, K. and Yangarber, R., Information-theoretic modeling of etymological sound change. In *Approaches to measuring linguistic differences*, Borin, L. and Saxena, A., editors, number 265 in Trends in Linguistics, Mouton de Gruyter, 2013, pages 507–532.

ZCF+10 Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. and Stoica, I., Spark: Cluster Computing with Working Sets. *HotCloud*, 10, page 10.

Appendix 1. Sample Output for Finnish

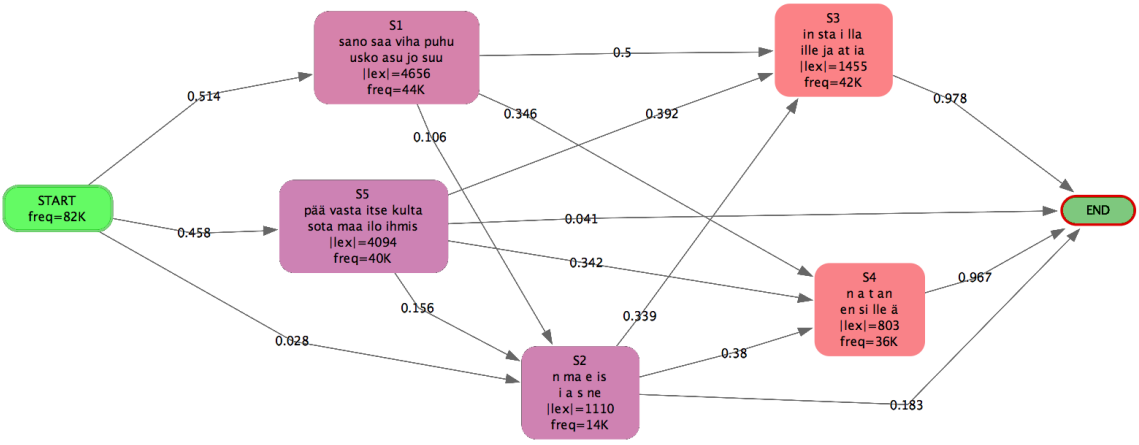

Figure 14: Visualization of a model with 5 states trained with Finnish data

A non-directional model trained with the Finnish data is visualized in Figure 14. The model is trained only with 5 classes for better visualization. The 8 most frequent morphs emitted from each state along with number of distinct morphs ($|Lex|$) and emission frequency ($freq$) is shown in each node. Probability of transition between the states is shown on the transition edges. Edges with very low transition probabilities are not shown in the graph. As can be seen, the model has learned to often emit stems from states S_1 and S_5, and suffixes from S_3 and S_4. As expected, the states corresponding to the stems are larger than the ones corresponding to suffixes. These two groups of states exhibit different properties: S_1 and S_5 have a more flat distribution, whereas S_3 and S_4 have few morphs with very high frequencies and many morphs with very low frequencies.

Output segmentations for 150 randomly selected words are listed below, followed by an extract of the output lexicon. Each segmentation is a list of morphs followed by the state the morph is emitted from. For example `‘‘ääntä 5 ä 4’’` is the segmentation for the word *“ääntää”*, with the morph *ääntä* emitted from state S_5 followed by the morph *ä* emitted from state S_4. Only 60 most frequent morphs of each state are listed for the output lexicon. Each state of the lexicon is formatted as a list of morphs and emission frequencies.

Output segmentations:

ääntä 5 ä 4
abadd 1 on 3
ajatel 5 le 2 eksi 4 kaan 4
aleksandri 5 asta 3
apuna 5 nsa 4
ase 5 ka 3
atla 5 s 2 puku 2
avona 1 isesti 4
bosra 1 lle 4
et 1 lä 2
ev 1 il 2
fanny 5 lle 4
halu 1 ttomasti 3
harjoit 1 uksesta 3
hellä 5 sydäm 2 isten 4
herja 5 amaan 3
hieta 5 iselle 3
hoita 1 vat 3
huoka 1 isten 4
huolehti 5 kaa 3
huoli 1
huomaa 1 matta 3 ni 4
hyökkä 5 sin 3
irtautu 1 u 3
itse 5 stämme 3
jaakob 1 issa 3
jalust 5 ime 2 en 4
jis 5 mak 5 ja 3
jo 1 kin 4
joutila 1 iksi 3
käänty 1 isimme 3
kärkky 1 vät 4
käy 1 mäl 2 öitä 4
käyttä 1 äksesi 3
kaali 5 n 4
kaarr 5 uttaa 3
kallionkielu 2 n 4
kanna 1
kannu 1 stettu 3
kansa 1 ltani 3
karsi 1 ttiin 3
kauhu 1 si 4
kedo 5 illa 3
keitto 5 s 4
kekäle 1 istä 3
kerto 1 ikin 3
kertom 5 uksista 3
kesä 5 huoneen 4
ketun 5 häntä 2
kiiva 1 s 4
kirja 5 sta 3
kivääri 5 n 2 perä 2 llä 3
kohtu 5 nsa 4
kokoon 5 kutsu 1 miseen 3
kokoon 5 nut 3 tiin 4
konst 5 i 3
korva 5 isi 3
koura 1 n 4
kov 1 inkin 3
kuninkaa 1 llemme 3
kuninkaa 1 llista 3
kuoll 1 akseen 3
le 5 isku 5 na 4
leikkaa 5 mattom 2 issa 3
lepuu 5 ta 4
lii 5 ska 2 ksi 4
lika 1 kuoppaa 2 n 4
lopet 1 tamaan 4
maala 1 ri 4
mahtava 5 mmaksi 3

meero 5 min 4
menesty 1 s 4
muka 5 iseen 4
murtautu 1 isivat 3
neljä 5 stä 4 toista 4
neljä 5 ttä 4
neste 5 ttä 4
neuvo 1 tellut 4
neuvos 1 kunta 2 nsa 4
nime 1 en 4
nuorukais 1 ena 3
nystyr 5 ä 4
odotta 1 isinko 2
ohra 5 n 4
olem 1 us 4
orji 1 ensa 4
pääs 1 tyämme 2
paheks 1 umista 3
palo 1 kunnan 3
parta 5 an 4
pataljoon 1 ianne 3
piuk 5 alla 3
polje 5 tte 3
puhel 5 tiin 4
puku 1 namme 3
punertava 5 ssa 4
puoli 5 pyörry 1 ksi 2 ssä 3
pysty 5 isivät 3
ramp 5 oja 4
rauen 1 neet 3
rauke 1 n 2 ivat 3
riippuva 1 in 3
riita 5 a 4
riko 1 mme 4
ruhtinattar 1 enkin 3
ryöst 1 etty 3
ryöstettäv 1 i 2 ksensä 3
säär 5 e 2 stänsä 3
saav 1 athan 3
samalla 5 ista 4
sanel 5 i 3
sanoma 5 sta 3
sapatin 1 päivä 3
selom 1 in 3
seurust 1 el 2 ko 4
sidkia 1 lle 4
sinetöi 1 tiin 4
sive 1 ästi 3
sonni 5 mäellä 4
sukupolv 5 issa 3
sy 5 kin 4 tään 3
syöks 1 it 3
syntiuhri 2 ksi 4
tä 2 ltä 3
törmä 1 sana 2 inen 4
talo 1 onsa 3
tamaan 2 it 3
tanssiais 5 issa 3
tekijä 2
tiehe 5 ni 4
todistaja 5 mme 4
toist 5 enkin 3
tot 1 uuteen 4
tutu 5 t 4
tykki 5 mies 4
ulv 1 ahti 3
uupu 1 a 4
vaarn 5 a 4
vala 5 lla 3
valhe 5 kynä 2
valjast 1 utti 3
valkea 5 ta 4

valt 5 oineen 3
vanh 1 us 4
vapaut 1 tamme 2
vapise 5 tpa 3
varsin 5 a 2 isia 4
velvoi 5 ttaisi 4
verta 1 uksensa 3
virka 5 s 4

Output lexicon:

State 1:

sano 149
saa 101
viha 92
puhu 91
usko 90
asu 81
jo 79
suu 78
isä 75
istu 74
neuvo 71
osa 71
surma 71
kiro 69
tuo 69
vaimo 69
seura 68
aja 66
jalo 66
toivo 65
vaiva 64
halu 63
lähte 63
ot 63
kutsu 62
astu 60
kysy 59
käy 59
velje 59
anta 58
jumala 58
kuul 58
kuva 58
elä 57
katso 57
kiusa 57
voima 57
kauhistu 54
suru 54
syö 54
kansa 53
laske 53
muut 53
hedelmä 52
herra 52
silmi 52
talo 52
kirjoit 51
kuulu 51
lausu 51
maja 51
miele 51
nime 51
oma 51
tahto 51
tapa 51
tuho 51
sydäme 50
etsi 49
lapse 49

State 2:

n 340
ma 264
e 249
is 241
i 207
a 188
s 177
ne 168
tu 151
o 139
ta 130
ti 120
tele 116
llis 106
el 103
u 90
ko 83
ty 80
la 77
te 77
kka 71
tä 68
ka 66
na 64
ksi 60
us 60
in 57
mi 57
hi 55
ku 54
uksi 54
aja 52
ha 51
mise 51
tta 51
mattom 49

mä 48	taja 42	puole 38	tel 36
se 48	to 41	kon 37	de 35
mis 47	va 40	li 37	imm 35
ttä 47	em 39	utu 37	
kko 45	ile 39	nne 36	
ri 45	lä 39	pu 36	
ra 42	tte 39	re 36	

State 3:

in 1692	nut 319	koon 222	ani 165
sta 1035	iin 312	e 218	lleen 160
i 863	isen 312	asta 217	staan 160
lla 729	maan 303	taan 206	usta 158
ille 586	iksi 302	va 196	alle 155
ja 469	on 298	isin 191	taa 153
at 431	ssä 298	jen 191	aksi 148
ia 402	ivat 286	nyt 184	ilta 145
illa 390	it 271	ina 183	amme 143
isi 384	ten 259	u 183	teli 143
neet 379	istä 249	alla 182	ttiin 143
vat 378	ita 248	iä 178	essa 142
et 372	tte 247	ttu 178	
ansa 355	kaa 245	asi 172	
issa 338	aa 232	uksen 171	
llä 319	ti 229	tä 168	

State 4:

n 2697	ssa 761	ta 412	us 272
a 2273	ista 727	s 408	nsä 268
t 1482	nsa 701	stä 400	än 257
an 1232	ksi 653	nne 398	iset 250
en 1182	ni 621	na 361	vät 217
si 893	kin 609	lta 353	kaan 194
lle 873	mme 596	ko 320	ihin 187
ä 798	inen 539	aan 316	assa 186

kö 171
ensa 167
han 164
van 163
nä 162
tti 156
sti 152
mään 149
isia 140
kseen 140
iden 139
kään 139
isten 137
eni 133
änsä 133
illä 125
issä 124
ensä 115
eksi 113
elle 113
ineen 110
hän 106
telee 104
ellä 103
iseen 103
vä 103
pa 102
esta 97

State 5:

pää 120
vasta 108
itse 106
kulta 104
sota 101
maa 97
ilo 95
ihmis 93
suur 87
kivi 84
turva 76
metsä 74
käsi 72
työ 70
juhla 67
silmä 67
hopea 66
matka 65
kunnia 64
määrä 64
tuli 64
armo 62
puu 62
arvo 60
si 60
sivu 60
liha 59
nuor 59
sala 59
väli 59
karja 58
vesi 58
puna 57
vaski 57
ala 56
jalka 55
suku 54
vara 54
men 53
se 53
tietä 53
valta 53
hel 52
kirja 52
mieli 52
otta 52
peri 52
päivä 52
uhri 52
temppeli 51
peit 50
ratsu 50
taistelu 50
taka 49
ase 48
ku 48
rinta 48
etu 47
korva 47
koti 47