THE UNIVERSITY OF CHICAGO

MORPHOLOGICAL PARADIGMS: COMPUTATIONAL STRUCTURE AND UNSUPERVISED LEARNING

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE HUMANITIES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF LINGUISTICS

BY
JACKSON LUN LEE

CHICAGO, ILLINOIS
MARCH 2022

To my parents

The strongest requirement that could be placed on the relation between a theory of linguistic structure and particular grammars is that the theory must provide a practical and mechanical method for actually constructing the grammar, given a corpus of utterances.

— Noam Chomsky, *Syntactic Structures* (1957/2002), pp. 50–51

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This dissertation would not have been possible without John Goldsmith, who one summer asked me to look into what this work calls the "paradigm alignment" problem and has embodied the kind of scholar I strive to be ever since.

The linguistics community at the University of Chicago was a vital part of my scholarly training. I am grateful to Alan Yu and Jason Riggle for their guidance through my dissertation research. I also benefited tremendously over the years from other UChicago linguistics faculty members not on my dissertation committee, especially Diane Brentari, Greg Kobele, and Salikoko Mufwene. Many hours were spent with my cohort: Andrea Beltrama, Tasos Chatzikonstantinou, Mike Pham, and Diane Rak.

In addition to Linguistics, other units at the University of Chicago also played an indispensable role in my graduate education: Computer Science, Statistics, Psychology, and the Research Computing Center.

Stephen Matthews, my undergraduate mentor, introduced me to the field of language acquisition, which has become a key component in my research program.

I thank my family back in Hong Kong for their endless support. And I thank Caroline Frazia, my other half.

ABSTRACT

This dissertation develops an algorithmic approach to linguistics through the study of topics in unsupervised learning of linguistic structure related to morphological paradigms. This work emphasizes reproducibility, accessibility, and extensibility in linguistic research.

The first major chapter studies stem extraction, focusing on analyzing morphological paradigms one at a time. Given a morphological paradigm, what is the stem, and how can we tell algorithmically? While it might appear trivial to extract "jump" from the English verbal paradigm jump-jumps-jumped-jumping, any non-concatenative morphology in any language presents an immediate challenge to an algorithm based on a substring approach to stem extraction. From the perspective of minimizing description length, the stem is best modeled as the longest common subsequence across word forms in a given morphological paradigm.

The next chapter explores paradigm similarity, considering multiple morphological paradigms at a time. The linguistic phenomenon of interest is inflection classes. Cross-linguistically, inflection classes tend to exhibit partial similarity. For instance, while Spanish verbs are customarily categorized as -ar, -er, and -ir verbs, the -er and -ir verbs are conjugationally more similar to each other than either to the -ar verbs. This chapter develops a hierarchical clustering algorithm that characterizes such partial similarity across morphological paradigms in a tree structure.

The final major chapter explores how tables of morphological paradigms can be learned from raw data, such as an unannotated text corpus. The point of departure is Linguistica (Goldsmith 2001). While Linguistica induces morphological paradigms from a raw text by learning recurring morphological patterns called signatures, the relationship between signatures is unknown, which means signatures that differ by inflection classes are not connected. This chapter aligns the signatures from Linguistica by leveraging syntagmatic information available in a raw text corpus to induce what is akin to word category knowledge.

CHAPTER 1

INTRODUCTION

1.1 Introduction

Words in a language are related to one another in many ways. This dissertation is interested in one particular kind of relationship among words: morphological paradigms. Familiar examples include the English verbal paradigm for JUMP which has the four surface word forms of jump-jumps-jumped-jumping.[1] The focus of this dissertation is the surface word forms, which give rise to structure within and across morphological paradigms. This means that in general, our discussion of morphological paradigms abstracts away from language-specific morphosyntax and semantics. As will be clear below, this is necessarily the case when we are interested in unsupervised learning of linguistic structure, with the use of unlabeled and unstructured datasets.

While this dissertation is on natural language morphology and morphological paradigms, a central theme that ties all topics studied is how humans learn language. Linguists have long been interested in the problem of language acquisition. A primary goal of linguistics is to understand the grammar of the world's languages. Knowing how humans learn language is tantamount to understanding how grammar comes into being and why it is the way it is – precisely the goal of linguistics just mentioned. This dissertation falls within this general research area of language acquisition, and focuses on the learning of natural language morphology from a computational perspective.

In this introductory chapter, I contextualize the topics on morphological paradigms to be studied and discuss some foundational issues with regard to the kind of linguistic research for which this dissertation argues. First, in section 1.2, I introduce the topics studied in this dissertation and their connection. Then, I provide some background on morphological

1. By convention in the linguistic literature, words typeset in small capitals (e.g., JUMP) represent lexemes, i.e., as a shorthand to refer to a particular morphological paradigm.

paradigms in section 1.3. Finally, I discuss the significance of doing linguistic research in a computationally rigorous manner in comparison to traditional theoretical linguistics (section 1.4), and justify the importance of computational linguistics in terms of reproducible, accessible, and extensible research (section 1.5).

While this dissertation is dated 2022, much of the work reported was accomplished by late 2016.

1.2 Synopsis

This dissertation studies three topics with regard to morphological paradigms: (i) structure within a morphological paradigm, (ii) structure across morphological paradigms, and (iii) induction of morphological paradigms from unstructured data. The first two parts explore the questions we can ask when a list of morphological paradigms is given. The third part is a twist, and asks where morphological paradigms come from in the first place.

1.2.1 Stem identification: Structure within a morphological paradigm

To represent the morphophonological relationship between a regular English singular and its corresponding plural, let us suppose there is templatic knowledge of some form which might look like the following:

(1) /X/ SINGULAR ∼ /Xz/ PLURAL

(1) states the phonological relationship between regular singular-plural pairs in English: the two forms in a pair differ phonologically in that the plural form has an additional /z/ at the right edge (and the actual phonetic realization of /z/ for plurals follows its allomorphic distribution for [s, z, əz]).

Where does knowledge like (1) come from? It is reasonable to conjecture that a pattern of this form must have come from numerous singular-plural pairs:

(2) Some regular singular-plural pairs in English:

SINGULAR	PLURAL
table	tables
chair	chairs
book	books
church	churches
⋮	⋮

It is the pairs like those in (2) which give rise to generalized knowledge like (1) that enables one to think that for instance, 'wugs' is the plural form of the singular, nonce word 'wug'. How can knowledge akin to (1) be extracted?

In (1), 'X' represents the shared phonological material in the singular-plural pair. The ability to identify what this shared material is appears to be essential, although it might seem rather trivial for English singular-plural pairs where the crucial difference is the right-edge /z/ for plurality. When we think of English singular-plural pairs as morphological paradigms and broaden our consideration to other paradigms in and beyond English, such identification of the shared material in a given morphological paradigm is demonstrably more complex in formal terms and warrants much more thorough and comprehensive characterization. The common material in a paradigm, represented as 'X' so far, is what linguists call the *stem* of the morphological paradigm. Although identifying the stem for languages like English appears straightforward, many other languages display much more complex types of word structure. In Arabic, for example, the morphological paradigms for words related to writing has the stem "k-t-b" with non-continuous material, e.g., *katab*a 'he wrote', *yaktub*u 'he writes/will write'.

At first glance, languages like English look intuitively quite different from those like Arabic. They demonstrate the common distinction of concatenative and non-concatenative morphology, respectively. Researchers in both theoretical linguistics and computational lin-

guistics use drastically different terms and models to characterize these two types of morphology. If the goal of linguistics is to understand language as a whole, then there are no aprioristic reasons to assume that fundamental learning strategies—and their modeling—of stem identification should be different depending on the specific languages. The first part of my dissertation (chapter 2) focuses on the problem of stem identification, devising general and language-independent strategies for identifying stems and addressing long standing issues ranging across the nature of morphemes, linearity, and the diverse morphological types across the world's languages.

1.2.2 Paradigm similarity: Structure across morphological paradigms

In a morphological paradigm, if the stem is the shared material across all word forms, then what they do not share is the *affixal* material. Labels such as SINGULAR and PLURAL are morphosyntactic values of some given morphosyntactic feature (NUMBER, in this example with SINGULAR and PLURAL). As far as the English plural is concerned, there is general agreement that /-z/ is the default exponence for PLURAL; this is what is observed for the non-existent 'wug' as discussed. However, it is by no means the only morphological exponence of PLURAL in English. Other possible plural suffixes are the non-default cases, e.g., *goose∼geese, octopus∼octopi, ox∼oxen.* Although they are for the most part tied to lexically specific nouns, it is arguably possible that non-default plural morphology is productive; Bauer (2001, 3) provides examples of invented English nouns which might be subject to non-default plural morphology: *brox∼broxen, ceratopus∼ceratopi.* If we go beyond nouns, there is strong evidence that non-default morphology is extensible to non-existent words. Albright and Hayes (2002, 2003) propose a computational model which predicts English past tense forms and whose results correlate with laboratory-based behavioral data. For instance, they predict that there is non-trivial probability that the non-existent English verb *spling* has *splung* as the past tense form, and they show that experimental subjects did rate *splung* as

a plausible candidate. What do Bauer's point on English plurals and Albright and Hayes's empirical results on English past tense tell us? What appears to be non-default, unproductive morphology is far from being stagnant. On the contrary, knowledge of such morphology is constantly being called upon in a non-random way. This has much to do with structure across morphological paradigms, in connection to topics ranging across inflection classes and allomorphy.

On structure across morphological paradigms, the traditional descriptions and analyses of patterns across morphological paradigms deserve some remarks at this point. Focusing on inflection classes, we take Spanish verbs as examples. Spanish verbs are described in terms of three distinct inflection classes, namely the -AR, -ER, and -IR verbs. The verb HABLAR 'to speak' is an -AR verb and has word forms such as *habl<u>o</u>-habl<u>a</u>-habl<u>amos</u>* (I speak; you speak; we speak), with the suffixes underlined. Other verbs take different patterns, e.g., the -ER verb COMER 'to eat' with *com<u>o</u>-com<u>e</u>-com<u>emos</u>* and the -IR verb VIVIR 'to live' with *viv<u>o</u>-viv<u>e</u>-viv<u>imos</u>*. These three Spanish verbs—HABLAR, COMER, VIVIR—illustrate the kind of similarities and differences across morphological paradigms that this part of my dissertation models. *Hablo, como* and *vivo* share the *-o* suffix, which makes the three verbs alike. However, *Hablamos, comemos* and *vivimos* all have distinct suffixes. Crucially, *comes* and *vives* share the *-es* suffix to the exclusion of *hablas* with *-as*. These Spanish verbs show that inflection classes are *partially* similar to one another with partial overlapping patterns. Traditional descriptions and analyses of inflection classes focus only on the overall differences which the grammatical endings display, but ignore this type of partial similarities among them; see also Costanzo (2011) on similar observations. This part of my dissertation develops the notion of paradigm similarity, and fills the gap in the current literature for the fine-grained modeling of structure across morphological paradigms, encompassing both their similarities and differences.

1.2.3 Paradigm induction and alignment: Learning paradigms from raw data

So far, the discussion assumes the availability of some full paradigms as the source of training data (for the template of English singular-plural morphology as in (1), for example). From the perspective of machine learning, if one attempts to predict that 'wugs' is the plural of 'wug' on the basis of a list of English singular-plural pairs, then this is *supervised* learning: the list of singular-plural pairs are the training data which trains a learning algorithm capable of predicting 'wugs' given 'wug'. For English past tense forms, this is exactly what Albright and Hayes do; their model is trained by a list of pairs of English bare verbs and their past tense forms, which in turn predicts 'splung' to be one of the likely past tense forms for the non-existent verb 'spling'. Now, the big question here is: where does a full paradigm table come from?

By a "full paradigm table", I refer to a table of morphological paradigms where each row contains word forms from the same *lexeme* and each column contains word forms from the same morphosyntactic category. The table in (2) for English singular-plural nouns is an example. It is clear to us—and to toddlers acquiring their first language—that full paradigm tables are not readily available for free. They must come from the tremendous amount of accumulated linguistic experience, to the point where it is possible for a competent speaker to sit down and construct a full paradigm table seemingly without much effort. This dissertation sets out to ask a challenging question: is it possible to induce paradigm tables from unstructured data, such as a text corpus? This is an *unsupervised* learning task, in the sense that the problem is to induce paradigm tables from raw data only and nothing else, without reference to any a priori given paradigm tables or individual paradigms. What makes this task challenging is the well-recognized properties of lexical statistics such as the Zipfian distribution of words and the pervasive problem of incomplete paradigmatic forms in a corpus.

1.3 On morphological paradigms

Morphology is the study of word structure. This dissertation research is concerned with structure across words related in certain particular ways – words that participate in morphological paradigms (sometimes simply "paradigms" for short). A morphological paradigm is a set of morphologically related words of the same *lexeme*; a lexeme is an abstract linguistic unit of words related by meaning. It is often—but not always—the case that a lexeme consists of words sharing a great deal in terms of their surface forms. A standard example is the English lexeme JUMP with word forms of *jump, jumped, jumping, jumps*, which share the surface phonological material of "jump", and the English lexeme BE represents an example where word forms share little among them, with *be, is, am, are, was, were, been, being*.

Word forms in a morphological paradigm can be either inflectionally or derivationally related; no strict distinction is drawn in this dissertation (cf. Spencer (2013)). The distinction of inflection versus derivation is not controversial. Inflectional paradigms can be construed as a matrix of lexemes as rows and morphosyntactic features as columns, with each cell occupied by some word form. Derivational paradigms are quite different and appear to be less consistently structured in terms of the matrix analogy for inflectional paradigms, for there are issues of world knowledge and productivity, to just name two. In computational morphology (see Goldsmith et al. 2017), there is a general tendency that many systems for automatically learning morphology focus on inflectional morphology, although they also learn derivational morphology, albeit to a much lesser extent. For example, it is entirely possible that a morphological learner concludes that both -s (inflectional) and -tion (derivation) are suffixes for a sizeable English dataset. As input datasets are usually void of semantic representation or world knowledge, this behavior of learning mostly inflectional morphology and some derivation is expected.

Morphological paradigms are considered central objects of study (Matthews, 1972; Bybee, 1985; Carstairs, 1987), in line with research under the rubric of Word and Paradigm Mor-

phology (Hockett, 1954; Blevins, 2013). Various paradigm uniformity and analogical effects (see papers in Downing et al. (2004)) speak for the position that morphological paradigms must be a legitimate category in the study of morphology that should receive serious attention from the linguistic research community. This contrasts with approaches to morphology that treat paradigms as epiphenomenal, e.g., syntax-based frameworks such as Distributed Morphology.

Languages can be classified in terms of morphological types. The traditional taxonomy—one that has been developed by and inherited from as far as the von Schlegel brothers in the early 19th century, and through Wilhelm von Humboldt and August Schleicher to Edward Sapir's *Language* (Sapir, 1921) and contemporary introductory linguistics texts—can be said to distinguish languages by the correlation among form, meaning, and wordhood. The widely recognized morphological types are isolating, fusional, agglutinative, and polysynthetic languages. As hypothetical and extreme types of languages, isolating languages have both lexical and grammatical morphemes as individual words, whereas polysynthetic languages are just the opposite, with a large number morphemes in a word that would correspond to sentences in English-type languages. The focus of this dissertation is morphological paradigms, particularly those in fusional and agglutinative languages whose morpheme-to-word ratio is in-between those of isolating and polysynthetic languages. On the one hand, we would like to work on non-isolating languages where word forms overtly contain both lexical and grammatical information, and on the other, we are also interested in the syntactic distribution of word forms and need relatively short words available in non-polysynthetic languages.

One reason why morphological paradigms are of great interest is their strong flavor of unseenness tied paradoxically with predictability. The particular kind of morphology under study in this dissertation research is not just words that are actually heard or written, but also those that are not. No matter how large a corpus of naturally occurring language is,

we are certain to find morphological paradigms for which not all possible word forms are observed in the corpus. While English verb lexemes have maximally five distinct word forms (*take, takes, taking, took, taken* for TAKE), a Spanish verb lexeme has over 50 distinct forms due to combinations of tense (present, past, and future) and mood (indicative, subjunctive, etc.) – it is hard to imagine a Spanish verb lexeme has all its word forms observed in a corpus. In other languages such as Basque and Finnish with even more complex morphology, the number of distinct forms of a single verb lexeme is in the order of hundreds, or even millions for Archi (Kibrik, 1998). Building on previous work, this dissertation develops a comprehensive computational system that induces morphological paradigms from a raw text and predicts unobserved paradigmatic forms.

1.4 Doing linguistic research computationally

Beyond morphological paradigms, this dissertation has the general goal of exploring and making explicit what a linguistic analysis is about. In the following, I elaborate in section 1.4.1 on the connection between this question and this dissertation, and argue in sections 1.4.2 and 1.4.3 for the computational perspective taken in this dissertation.

1.4.1 The relationship between data and analysis

In linguistics, what does it mean to come up with an analysis for some data? This section reflects upon this by discussing two seemingly incompatible positions as an answer to it and providing my view that reconciles the apparent paradox.

The first position for how a linguistic analysis comes into being is the more traditional approach. In theoretical linguistics, this actually goes by and large unnoticed. Given data, the job of a linguist is to provide the best analysis that captures the full range of the available data. This is all too familiar to those who have undertaken graduate-level linguistic training in the field, in the sense that when you are given a linguistics problem set and asked to analyze

the data, your job is to come back a while later with an analysis—the *best* analysis—in your write-up. Nobody would ask exactly what enables you to know the analysis and what has happened in you and your brain between the moment you receive the problem set and the moment you submit the analysis. What goes between the data and analysis is very much like a black box. This first mode of linguistic research can be represented by this diagram:

(3) Traditional approach to linguistic research

$$\text{data} \rightarrow \text{linguist} \rightarrow \text{analysis}$$

The second position for doing linguistics can be regarded as "the algorithmic approach". Most importantly, what mediates between the data and analysis is an algorithm or procedure of some sort, entirely external to the human mind and, once established, free from any human biases. In the mode of doing linguistic analysis, the human analyst still has access to both data and analysis, but restricts direct access and manipulation to only the algorithm that takes the data and outputs the analysis. If the analysis is suboptimal (as evaluated by the analyst), then changes can be made only to the algorithm. At any given point, the algorithm can be externally and objectively examined. Given its nature of absolute explicitness, the algorithmic approach to linguistics is usually practiced by the computationally oriented researchers who implement algorithms as computer programs. This second mode of linguistic research can be represented by the following diagram – note how it contrasts with the previous diagram above:

(4) An algorithmic approach to linguistic research

$$\begin{array}{ccccc} \text{data} & \rightarrow & \text{algorithm} & \rightarrow & \text{analysis} \\ & & \uparrow & & \\ & & \text{linguist} & & \end{array}$$

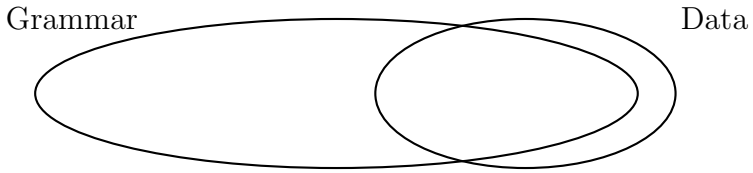
Prima facie, these two approaches to linguistics appear incompatible. Indeed, as will be clear, this dissertation shows the value of the algorithmic approach to linguistics, but it does

not mean that we should ditch the more traditional approach. It is important to recognize that, after all, the algorithmic approach to linguistics depends on the insights of the human analyst who develops an algorithm. Such insights do stem from the human intuition. Also, whether an algorithm is deemed successful relies on some evaluation metric which is devised, necessarily, based on the human analysts' judgment as to what it means to be a good algorithm or analysis. My dissertation shows that theoretical linguistics stemming from human insights and computational linguistics built on rigorous implementation complement each other, and that neither of them is dispensable for the explicit modeling of the relationship between data and analysis.

1.4.2 Empiricism, language acquisition, and machine learning

I take a strongly empirical view of language and linguistic research. More specifically, my view is based on the rather uncontroversial position that the relationship between grammar and data is one that is partially overlapping:

(5) The partially overlapping nature of grammar and data



The term "data" refers to what is objectively and empirically observed in the real world, i.e., what one actually hears and utters. "Grammar" refers to what an individual knows about a particular language. The view of the partial overlapping between grammar and data is uncontroversial, as depicted here, in the following sense. On the one hand, there is something which we have never heard or uttered but which we know is part of the language in question, i.e., in the grammar but not in the data. On the other hand, there is a small portion of the observed data that can be considered the noise in the data, analogous to

errors, slips of tongue, or anything that goes under the heading of "I heard it before but I'd never say that" – this is part of the observed data but not in one's grammar.

Arguably, mainstream linguistic research focuses on what is referred to as "grammar" above. Linguists are interested in characterizing what is *not* observed and yet considered part of grammar. This is reflected by the widespread use of the introspection methodology leading to grammaticality judgments (often from by the authors themselves) that advance theoretical arguments. A related and increasingly popular methodology is to conduct behavioral studies to obtain data from a much larger pool of subjects who are native-speaker consultants for more fine-grained grammaticality judgments.

While one of the most intriguing aspects of language is our implicit knowledge of grammar, it is important to recognize that grammar ultimately comes from data. This point is especially true for morphology – for the wug-type knowledge described in section 1.2 above. This dissertation asks how knowledge of this sort can be acquired from data and modeled computationally, and therefore argues for a strongly empirical and learning-focused approached to language and linguistic research. A research program on how grammar results from data echoes the recent literature on learnability, especially regarding the observation that most mainstream theoretical linguistic work focuses on characterizing the grammar and ignores questions of how it comes into being (cf. Clark 2015). In addition, focusing on language acquisition and learning is also what will ultimately shed light on other challenging aspects of language, such as variation and change which are intimately connected to learning.

From this perspective, my dissertation research is generally couched within the area of language acquisition. How do humans acquire language? How do we go from nothing to something, as it were, from real-world linguistic data to an abstract grammar? The view that there is an abstract grammar is strongly supported by the wug-related observations and productivity. This dissertation shows that we gain insights about acquisition by studying language from the perspective of unsupervised learning.

By unsupervised learning, I refer to the ensemble of computational tools and concepts from computer science and statistics employed in learning patterns from unlabeled data. The criterion of using unlabeled data distinguishes unsupervised learning from supervised learning which, by definition, relies on the availability of training data. All else being equal, supervised learning strategies lead to better results. In the area of natural language processing for applied and engineering purposes, this is desirable for practical reasons, so long as training data are available. In contrast, unsupervised learning has its own importance for different reasons. Particularly relevant in the context of this dissertation is the use of unsupervised learning techniques as a way to model first language acquisition (Clark and Lappin, 2010), given that toddlers acquiring their first language have to induce an abstract grammar based only on the unlabeled linguistic data from the ambient environment. While using unsupervised learning techniques to model a linguistic property or phenomenon might lead to poorer accuracy compared to supervised methods, unsupervised learning comes closer to the real-world linguistic scenario, where there is no gold standard neatly prepared (i.e., training data with labeled answers).

For language acquisition, the relevance of unsupervised learning for morphological paradigms is clear from the discussion above on the wug problem with a raw text as the starting point. In authentic conversations and texts, there is never a full paradigm table available in front of a child acquiring their native language. Compounding the apparent difficulty of learning morphological paradigms are the problems of insufficient positive evidence and lack of negative evidence: a child does not typically encounter all word forms of all possible paradigms, and they are usually not explicitly corrected when they make an error. Such apparent empirical challenges in language acquisition are part of the Poverty of the Stimulus argument (Chomsky, 1965) for the nativist view about language. The dual problem of no negative evidence and insufficient positive evidence in naturally occurring linguistic data will have to be overcome in unsupervised approaches of language acquisition.

1.4.3 Grammar evaluation and algorithms

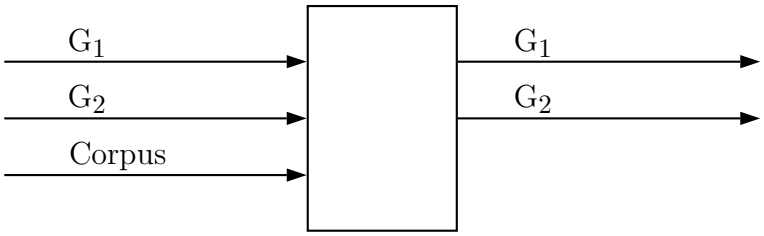
Criss-crossing this dissertation is the question of what it means for an analysis to be the best for some given linguistic data. Linguists seem to share the intuition—be it vague or not—for what makes an analysis good or bad, including such factors as how complex the formalism is and how much data the analysis can capture; see Halle (1962) on measuring grammar complexity in terms of counting symbols and Bochner (1992) specifically on morphology. This dissertation pursues the view that much as we have to be explicit about *what* it means for an analysis to be the best, it is no less important that we are explicit about *how* we advance an analysis. The "what" question is discussed in terms of the trade-off between grammar complexity and data compression, whereas the answer to the "how" question relies on the use of computationally implementable algorithms; both parts are closely connected.

On the goal of linguistic theory, Chomsky (1957/2002, ch.6) discusses several possibilities and argues for this particular one:

> "[G]iven a corpus and given two proposed grammars G_1 and G_2, the theory must tell us which is the better grammar of the language from which the corpus is drawn. In this case we might say that the theory provides an *evaluation procedure* for grammars." (Chomsky, 1957/2002, 51; original emphasis)

This is graphically represented on the same page as follows:

(6) Chomsky (1957/2002, 51) on the evaluation procedure of grammars



Early responses to this position appear to be somewhat cautious, but by no means negative. Garvin (1964, 36-37) wrote:

"I do not wish to become embroiled in the ultimate issue raised, namely, whether an evaluation procedure is indeed the only reasonable goal for linguistics. In my opinion it is not.

I should instead like to discuss a more immediate question: how to provide a practical evaluation procedure, where by practical I mean something that can actually be done in practice.

It appears to me first of all that an evaluation procedure for grammars – that is, entire grammars – is rather a tall order, if the procedure is to be interpreted operationally.

It seems to imply that two grammars (that is, two complete descriptions from phoneme – or morphoneme [sic] – to sentence) are to be compared to each other and to a corpus, in order to ascertain which is to be preferred. As to the criterion by which this is to be judged, let me quote Chomsky again: "Suppose that we use the word 'simplicity' to refer to the set of formal properties of grammars that we shall consider in choosing among them." For the choice to be feasible in practice, there should first of all exist two such grammars, each meeting what Chomsky calls "the external criteria of adequacy for grammars". Assuming such a condition, the two adequate grammars would then have to be compared in their entirety which, if taken seriously, might mean a comparison page by page, or statement by statement, or chapter by chapter. Each partial comparison may then result in a judgement of simplicity. If it is possible to weight each part judgement appropriately, one may assume that an overall judgement can be computed by some reasonable statistical operation. It is also thinkable that instead of this series of partial comparisons (which presupposes a matching of parts that are not necessarily susceptible to clear-cut matches), one might take each grammar separately and by some procedure to be defined when available

> take an independent measure of simplicity. The two measures could then be compared and a final evaluation made.
>
> I wonder whether, at the present state of the art in linguistics, any of this is very practical.
>
> If an evaluation procedure for entire grammars is deemed impractical, under what conditions does an evaluation procedure in linguistics become practical, and is it then relevant to the objectives of linguistics? (Garvin, 1964, 36-37)

Although Garvin's remarks quoted above may seem less than positive, they are concerned with *entire grammars*. He actually went on—with a much more positive tone—showing how grammar evaluation might be done for a subpart of grammar; as it happens, his example is about morphological analysis. Like Garvin, we attempt to explore and implement evaluation in linguistic analysis. If this is the way to do linguistics, the question we ask now is this: How exactly do we evaluate which grammar or analysis is the best among competing ones? The answer we adopt here is to incorporate the Minimum Description Length philosophy (MDL; Rissanen 1989) in an algorithmic approach. There are two parts here, which have been argued for in linguistics, see Goldsmith (2011a) on MDL, and Goldsmith (2004) on algorithms.

For any grammar or linguistic analysis, we would like a way of formalizing how complex it is. In the parlance of computer science, complexity can be formalized as description length in terms of bits: the longer the description length, the more complex the analysis is. Furthermore, we also ask how good the analysis is for fitting the given data, and for this we measure it by the number of bits needed to encode data using a particular grammar. The notion of the best analysis, then, is formalized as searching for the one with the smallest sum of the two measurements. This is in line with the philosophy of MDL approaches in machine learning. In other words, for grammar selection, an MDL analysis asserts that the best analysis is one which minimizes the sum of the grammar complexity and the data cost given

the grammar (Goldsmith, 2011a). There are two major appealing aspects stemming from an MDL approach. First, MDL embodies Occam's razor. Minimizing grammar complexity is the computational analog to advancing the simplest analysis in theoretical linguistics. Second, MDL eschews over-fitting. In traditional theoretical linguistics, an often unnoticed assumption is the emphasis placed on accounting for *all* the given data points at all costs, typical in linguistic training, and consequently the reduced concern over increasing grammar complexity. In a nutshell, the insight from MDL is this: we do not want to fit the data too well at the cost of a highly complex grammar, and at the same time we also do not want a grammar that is too simple, one that fits the given data too poorly. MDL says that the best analysis is a trade-off between how complex the analysis is, on the one hand, and the goodness-of-fit by that analysis for the given data, on the other.

On algorithms, we show the value of using them as they provide a computationally explicit and falsifiable means to derive and test analyses. More often than not, in theoretical linguistics, the focus is the analysis but not *how* that analysis comes into being in the first place. This claim is supported by the way in which linguists are typically trained: given a linguistic dataset, we are asked to come up with an analysis for a given question, but we are *never* asked to explicitly and meticulously pin down the steps through which the analysis comes into being. The procedure which leads to an analysis is as important as, if not more, the analysis itself (Goldsmith, 2004). Such a procedure is an algorithm. An algorithmic approach is especially relevant in the computer age. With the high computational power right at our fingertips, an algorithm can easily run through a huge amount of data, perhaps from different languages. We shift our focus to the procedure resulting in an analysis and to the interpretation of the analysis. An important advantage of algorithmic approaches to linguistic analysis is how they potentially shed light on questions of language learning. An algorithm in the context of this dissertation research is the computational implementation of an explicit model of precisely how linguistic generalizations are learned for some given data.

1.5 Axioms of computational linguistic research

My doctoral research reflects my commitment to reproducible, accessible, and extensible research. Whereas the value of reproducible research is widely recognized across the scientific community, I argue that it is insufficient for linguistics: linguistic research must also be accessible and extensible. In order that linguistic research be reproducible, accessible, and extensible, I also argue that this goal is concretely achievable by constructing user-friendly and freely available computer software.

1.5.1 Reproducible research

Reproducible research refers to the idea that the publication of academic research is the ensemble of published papers together with all datasets and tools (experimental stimuli, computer code, etc.) which produce the reported results. The notion of reproducible research, especially for computationally oriented research like this dissertation, was first articulated by Claerbout and Karrenbach (1992) in the field of geophysics. They went as far as suggesting that a publication should be an electronic document which is coupled with all the relevant software and datasets, and which have buttons that a reader can click to regenerate all analyses and figures. While technologies have advanced between now and then, the version of reproducible research I describe in this dissertation is along the lines of Claerbout and Karrenbach (1992) in spirit as well as congruent with the current technical standards. The very high bar set by Claerbout and Karrenbach is unlikely to be generalizable and applicable to multiple fields. Publication of scientific articles has remained the major focus in academia, as is the case for linguistics. A viable option for reproducible research in linguistics, at least given the present-day circumstances, seems to be that authors can point to a website with code and datasets together with clear instructions for how to rerun everything. This is arguably not ideal, but until a universal solution comes, it is the author's responsibility for maintaining the availability of their materials online.

Note that our focus here is *reproducible* research, as opposed to *replicable* research. A key difference is that reproducible research is about using the exact same dataset and code to regenerate results, whereas replicable research is about collecting data afresh by following a previous study and attempting to achieve comparable main results. In linguistics, therefore, replicable research is more amenable to experimental work, though if the researcher makes the experimental dataset publicly available, it will allow reproducible research for re- and meta-analysis. Relatedly, Mark Liberman's Language Log blog post "Replicability vs. reproducibility – or is it the other way around?" discusses their distinction and how they might be confused.[2]

There are a multitude of philosophical, academic, and administrative reasons why abiding by the protocol of reproducible research is desirable. Here I mention two major reasons. First, reproducible research facilitates data reanalysis and comparison of results across researchers.

For a variety of reasons, however, reproducible research has not been a widespread practice, which has led to growing concerns in fields as diverse as psychology (Wicherts and Bakker, 2012), pharmacology (Prinz et al., 2011), and computational engineering (Mitchell et al., 2012). For linguistics, recent works which express such concerns include Pedersen (2008) on computational linguistics and Maxwell (2012) on grammar descriptions. The open access online journal *Journal of Experimental Linguistics*, as part of the Linguistic Society of America's eLanguage initiative, represents an effort of encouraging reproducible research in the field.[3]

Given the specific circumstances of linguistics, reproducibility alone is insufficient for linguistics. For linguistic research to be practically and meaningfully reproducible, it must also be accessible and extensible.

2. `http://languagelog.ldc.upenn.edu/nll/?p=21956`, accessed in June 2016

3. `http://elanguage.net/journals/jel`

1.5.2 Accessible research

Accessible research is the idea that research should be relatively accessible by the practitioners in a given field. This is significant for linguistics. For historical and institutional reasons, linguistics departments at universities are usually located within social sciences or humanities; at universities without a linguistics department, linguistics is at the English department or a foreign language department. Given the general research methodologies in social sciences and (even more so) in humanities, it is both unrealistic and unreasonable to expect scholars in these areas to possess a high level of computing skills (e.g., for operations with computer scripts and tasks involving the command-line interface alone). Being able to communicate one's research—however technical it is—to other practitioners in the field is among our priorities. To achieve this, a clear direction is to create user-friendly computational tools.

Another component of accessible research for linguistics is data visualization. On the one hand, the focus of linguistic research has shifted from categoricity to gradience, thereby employing a wide range of experimental, quantitative, and computational methodologies. On the other, parallel with the phenomenon of Big Data, linguists are increasingly dealing with datasets (experimental data, corpora, etc) whose sizes are ever-growing; it is no accident that the Linguistic Society of America's Summer Institute 2015 at the University of Chicago had the theme of "Linguistic Theory in a World of Big Data".[4] This trend requires tools to explore patterns and present information from linguistic data of huge sizes as well as high complexity and gradience.

1.5.3 Extensible research

Extensible research in linguistics refers to the idea that analyses should be amenable to extensions for both modularity and cross-linguistic coverage. If we study language in terms

4. `http://lsa2015.uchicago.edu`

of various subfields, then it must be the case that our knowledge about language from all the fields has to all come together in some integrated way as a general model of language. The fact that knowledge is studied in a segregated way in linguistics does not mask the empirical fact the language works as a whole. If the goal of linguistics is to search for a general understanding of how language works, then research from any linguistic subfields must have ramifications for and contributions to other subfields. This view is expressed most concretely by all kinds of linguistic interface research; see, for example, the papers in the edited volumes by Ramchand and Reiss (2007) as well as Folli and Ulbrich (2010). To make it possible for research to cross subfields given their compartmentalization, it is best for linguistic research to be modular with well-defined input and output points so that different research can be connected – this will be clear below with my dissertation research as an example.

The second aspect of extensible research is the familiar goal that general linguistic research be cross-linguistically relevant. Although many technical papers in linguistics are in-depth case studies of a very small number of languages, most of them attempt to convey the message that they have something to say about language in general according to the insights learned from their case studies. For non-computational research, to see how claims based on a few languages bear on other languages usually requires the same huge amount of work in terms of time, labor, and mental energy as the original paper where the claims come from. To the extent that this is a form of extensible research, computational work has the added advantage of being much more efficient to reproduce given suitably prepared datasets from other languages.

Reproducible, accessible, and extensible research can be achieved by the development of open-source and user-friendly software. In computational research both within and outside linguistics, open-source software with computer code publicly available allows reproducible research. Creating a GUI makes computational linguistic research easily accessible to more practitioners, both in terms of usage and data visualization, as well as extensible to other

languages and further work.

All components of my dissertation are associated with user-friendly software applications which linguists can use with easily prepared datasets. The verifiability and replicability of linguistic analyses are among the strong reasons why producing software is important; we need a consistent, objective, and accessible way of applying linguistic analyses to different datasets from different linguists and sources. Creating software and making it available to other analysts is precisely a concrete and convincing way to achieve this goal.

All the software, source code, and datasets that are used in connection with my dissertation research are hosted online.[5]

1.5.4 Linguistica 5

A significant effort of this dissertation research was devoted to the development of software packages, in line with reproducible, accessible, and extensible research. A focus point is *Linguistica 5* (Lee and Goldsmith, 2016a). Its linguistic results will be discussed in the relevant chapters. Here, I briefly note its software engineering aspects.

Previous versions of Linguistica (Goldsmith, 2001) are written in C++ and built in the Qt framework. These versions are designed to be GUI software out of the box. The major drawback is that the core backend is intimately tied with the GUI code, which makes further development and debugging challenging. To solve this problem, the new *Linguistica 5* takes a radically different approach.

First, we choose Python to be the new programming language for Linguistica, because it has been widely used in computational linguistics and natural language processing for its strengths in fast coding, strong library support for machine learning and other computational tools.

Second, the focus of the *Linguistica 5* development is its backend as a Python library,

5. `https://github.com/jacksonllee`

with a GUI wrapper written in PyQt. This new architecture has several advantages. In terms of the user interface, there are two independent choices. As in previous versions of Linguistica, the GUI allows convenient data analysis – and visualization, a new development in *Linguistica 5*. Another novelty is that *Linguistica 5* is a Python library by design. Researchers are able to use *Linguistica 5* in a computationally dynamic and automatic fashion by calling it in their own programs for any research and computational work of their interest.

CHAPTER 2

STEM EXTRACTION

2.1 Introduction

This chapter is about stem extraction. The goal is two-fold. First, this chapter characterizes structure *within* a morphological paradigm; the subsequent chapters go beyond a single morphological paradigm. Second, we develop unsupervised methods of stem extraction based on properties of a morphological paradigm.

We begin with a question that may look deceptively simple: Given a morphological paradigm, what can be said about it linguistically? An intuitive response is that most if not all the words in a morphological paradigm share some phonological material. This is illustrated by the following examples.

(7) Some morphological paradigms cross-linguistically

person.number	Spanish *hablar* "speak"	English *jump*
1.SG	hablo	jump
2.SG	hablas	jump
3.SG	habla	jumps
1.PL	hablamos	jump
2.PL	hablaís	jump
3.PL	hablan	jump

(7) shows two verbal morphological paradigms, one from Spanish and the other from English; these paradigms are all in their present tense form. For our present purposes, it is sufficient to treat orthography as an adequate representation of phonology.

(7) first shows the Spanish paradigm for HABLAR 'to speak' in present indicative. The six word forms in this paradigm are different but all share some phonological material. In

particular, we would say that they all share the material "habl" (not worrying about the phonetically silent "h" in Spanish). The second paradigm in (7) is the one for English JUMP. Similar to the Spanish HABLAR paradigm, all six word forms in the JUMP paradigm share some phonological material, and it is uncontroversially "jump". Statements such as these about "habl" being the shared material for Spanish HABLAR and "jump" for English JUMP are what this chapter attempts to understand. For Spanish HABLAR, what makes us claim that "habl" is what the six words share? Why would we not say it is "hab", for instance? We shall see in this chapter that what it means to be shared material among word forms is far from being trivial.

The observation that word forms in a morphological paradigm share a fair amount of phonological material makes it reasonable to ask if there are principled ways of extracting such common material—call it stem, following the linguistic literature—from a morphological paradigm, hence the task of stem extraction. Pursuing this task bears both theoretical and practical interests. On the one hand, we would like to develop a deeper understanding of a task that appears to be second nature to linguists performing morphological analyses. Identifying the stem in a paradigm entails that the affixes—what is *not* shared among the word forms in the paradigm—are also identified, a central component of morphological analysis. On the other hand, stem extraction finds practical applications in language technologies, e.g., as part of stemming and information retrieval implemented in databases and search engines.

To further illustrate the types of morphological patterns that stem extraction should be able to handle, there is non-concatenative morphology where the stem is not formed by contiguous symbols, e.g., the root-and-pattern morphology in Semitic languages. (8) shows an Arabic paradigm for "write", for which the stem is the non-contiguous k-t-b.

(8) Arabic "k-t-b"

'he wrote'	**k**a**t**a**b**a
'we wrote'	**k**a**t**a**b**nā
'he writes, will write'	ya**kt**u**b**u
'we write, will write'	na**kt**u**b**u
'writer'	**k**ā**t**i**b**
'he dictated'	a**kt**a**b**a
'he dictates, will dictate'	yu**kt**i**b**u

In addition, many languages have morphological paradigms of both contiguous and non-contiguous types of stems, contiguous and non-contiguous. Spanish COMER in present indicative, in (9), has the contiguous stem "com", whereas PODER, in (10), has the non-contiguous "p-d".

(9) Spanish "com"

I eat	**com**o
you (sg) eat	**com**es
he/she/it eats	**com**e
we eat	**com**emos
you (pl) eat	**com**éis
they eat	**com**en

(10) Spanish "p-d"

I can	**p**ue**d**o
you (sg) can	**p**ue**d**es
he/she/it can	**p**ue**d**e
we eat	**p**o**d**emos
you (pl) can	**p**o**d**éis
they can	**p**ue**d**en

To identify the stem in a paradigm, the intuition is that we wish to find the *maximal common material* across all word forms (Spencer, 2012). By drawing insights from mathematics and bioinformatics, I describe three language-independent and algorithmic approaches—substrings, submultisets, and subsequences—which define the notion of "common material", and conclude that the *subsequence* approach most closely matches what is desirable (see also Hulden et al. (2014)). Preliminary results were presented in Lee and Goldsmith (2016b).

2.2 Formal aspects

This section develops the formal background of representing morphological paradigms, drawing from formal language theory and standard views of morphological paradigms in linguistics.

2.2.1 Total and partial words

Following standard practice in formal language theory, we first define words. Then we extend words to partial words which allow unspecified symbols.

A word w is defined as a sequence of symbols from a non-empty finite alphabet $\Sigma = \{l_1, l_2, ..., l_k\}$ with k symbols (e.g., the English alphabet, the IPA symbols); for instance, the word "apple" is formally the sequence $\langle a, p, p, l, e\rangle$ whose length is 5. The set of all words in a language is $V = \{w_1, w_2, ...\} \subset \Sigma^*$, where Σ^* is set of all possible sequences over Σ. The (unique) empty string is designated as ϵ with its length $|\epsilon| = 0$.

On simple concatenation, the multiplicative notation is used. For instance, given $w_i = \langle a, b\rangle$ and $w_j = \langle c, d\rangle$, we write $w_i w_j = \langle a, b\rangle\langle c, d\rangle = \langle a, b, c, d\rangle$.

What we have been referring to as words can be more precisely be called *total words*, in contrast to *partial words*. Partial words are intuitively words that have unspecified letters. This section benefits from background on partial words in Blanchet-Sadri (2008) (**?**).

To start with, we define partial functions as well as holes.

Definition 1. *Let f be a function with the set X as the domain. If f is not necessarily defined for all $x \in X$, then f is a* **partial function**.

Definition 2. *Let f be a partial function with the set X as the domain. The set of* **holes** *H(f) in X is the elements in X for which f is undefined (denoted by $\uparrow$).*

That is, H(f) = { $x : x \in X$ and f(x) = $\uparrow$ }

We now consider a word to be a function mapping a set of indices to a set of letters from Σ. Indices are the set of non-negative integers {0, 1, 2, ...}. For instance, a word w could be intuitively "apple", and formally $w : \{0, 1, 2, 3, 4\} \rightarrow \{a, p, l, e\}$ defined as follows ($w(i)$ means the index i of w):

w(0) = a

w(1) = p

w(2) = p

w(3) = l

w(4) = e

In this example, w is a total function, where each element in the relevant domain (= each index) is mapped to an element in the codomain (= one of the symbols in $\{a, p, l, e\}$). Because w is a total function, there are no holes, i.e., $H(w) = \emptyset$.

We are now ready to define partial words.

Definition 3. *A* **partial word** *of length n over Σ is a function $w : \{0, 1, 2, ..., n-1\} \rightarrow \Sigma$ with $H(w) \neq \emptyset$. In other words, there exists at least one index i and w(i) = $\uparrow$.*

An example of a partial word is $v : \{0, 1, 4\} \rightarrow \{a, p, e\}$ defined as follows:

v(0) = a

v(1) = p

v(2) = $\uparrow$

v(3) = $\uparrow$

v(4) = e

Comparing v and w in these two examples, we can think of v as the word "apple" but with two missing symbols. While the concept of partial words allows us to refer to words with missing symbols, we would like to concretely represent partial words in a way analogous to how we usually think of words. We introduce two notations for unspecified symbols.

Definition 4. *The notation* $\lrcorner$ *represents one and only one unspecified symbol from* Σ*.*

Definition 5. *The notation* $\sim$ *represents a sequence* x *of unspecified symbols from* Σ*, where* $|x| \geq 0$*. That is,* $\sim$ *is a wild card for anything in* Σ^**.*

Note that both $\lrcorner$ and $\sim$ are not in Σ. Given these notations for unspecified symbols, v from above can be thought of as $\langle a, p, \lrcorner, \lrcorner, e\rangle$, in contrast with $w = \langle a, p, p, l, e\rangle$. Alternatively, v can also be represented as $\langle a, p, \sim, e\rangle$. Unspecified symbols are going to be important for the discussion on stem extraction.

2.2.2 Morphological paradigms: Stems, affixes, and paradigm sets

Loosely speaking, a morphological paradigm is a set of words in a natural language that are morphologically related. Both inflectional paradigms (e.g., jump-jumps-jumped-jumping) and derivational paradigms (e.g., create-creation) are relevant, although in this dissertation the discussion focuses on inflectional paradigms.

Formally, especially in the context of our discussion developed so far, a morphological paradigm is a set of words. Let us say this more concretely as a first step of defining a morphological paradigm:

Definition 6 (morphological paradigms, first version)**.** *A* **morphological paradigm** *is a set of words* $W = \{w_1, w_2, ...\} \subset \Sigma^*$*.*

We are interested in how morphological paradigms are connected to the notion of stems. To this end, we first need to define stems.

Definition 7. *A* **stem** *t is a sequence in $(\Sigma \cup \{\sim\})^*$.*

An example of stems which match linguistic intuition for English is $\langle j, u, m, p, \sim\rangle$ for the JUMP paradigm with the word forms jump-jumps-jumped-jumping, where $\sim$ in the stem is materialized as $\emptyset$-s-ed-ing.

Given a word, the part that is not the stem is the affix. Focusing on form, affixes are formally the same as stems.

Definition 8. *An* **affix** *a is a sequence in $(\Sigma \cup \{\sim\})^*$.*

A stem and an affix form a word, for which a word composition operation is defined.

Definition 9. *A* **word composition operation** $\oplus$ *takes a stem t and an affix a, and returns a total word $w \in \Sigma^*$.*

$w = t \oplus a$

The algorithmic details of $\oplus$ depend on what types of stems we are concerned with; more details on this in section 2.3 on stem extraction. As a quick illustration of what a morphological paradigm looks like in our set-up so far, consider the JUMP paradigm:

$$\left\{\begin{array}{c}\langle j,u,m,p\rangle\\ \langle j,u,m,p,e,d\rangle\\ \langle j,u,m,p,i,n,g\rangle\\ \langle j,u,m,p,s\rangle\end{array}\right\} = \langle j,u,m,p,\sim\rangle \oplus \left\{\begin{array}{c}\langle\sim\rangle\\ \langle\sim,e,d\rangle\\ \langle\sim,i,n,g\rangle\\ \langle\sim,s\rangle\end{array}\right\}$$

Here, we assume that $\oplus$ can be applied distributively for one given stem with multiple affixes to produce the corresponding (total) words.

A revised definition of a morphological paradigm is as follows:

Definition 10 (morphological paradigms, second version)**.** *A* **morphological paradigm** *is a set of words $W = \{w_1, w_2, ...\} \subset \Sigma^*$ such that there exists a stem t, a set of affixes $A = \{a_1,\ a_2,\ ...\} \subset (\Sigma \cup \{\sim\})^*$, and $W = t \oplus A$.*

What counts as a stem requires further restrictions. For instance, we would not want to claim that English table-tired-toy is a paradigm sharing the stem "t"; meaning could certainly be part of the restrictions. If we focus on form, it is helpful to speak of a lower bound on the number of stems compatible for a given set of affixes. This leads us to *paradigm sets*.

An important reason why we say that jump-jumped-jumping-jumps is a paradigm in English is because there exist *other* paradigms with the same affixes: the WALK, TREAT paradigms are among the examples. In other words, implicitly, when linguists speak of a morphological paradigm, what is also being referred to is a *paradigm set* in the background: the JUMP paradigm is a paradigm due to the paradigm set consisting of JUMP and other verb paradigms with a similar affix pattern of ∅-ed-ing-s. A paradigm set with two paradigms is as follows:

(11) A paradigm set with two paradigms

$$\left\{ \begin{Bmatrix} \langle j,u,m,p \rangle \\ \langle j,u,m,p,e,d \rangle \\ \langle j,u,m,p,i,n,g \rangle \\ \langle j,u,m,p,s \rangle \end{Bmatrix} \begin{Bmatrix} \langle w,a,l,k \rangle \\ \langle w,a,l,k,e,d \rangle \\ \langle w,a,l,k,i,n,g \rangle \\ \langle w,a,l,k,s \rangle \end{Bmatrix} \right\} = \begin{Bmatrix} \langle j,u,m,p,\sim \rangle \\ \langle w,a,l,k,\sim \rangle \end{Bmatrix} \oplus \begin{Bmatrix} \langle \sim \rangle \\ \langle \sim,e,d \rangle \\ \langle \sim,i,n,g \rangle \\ \langle \sim,s \rangle \end{Bmatrix}$$

If we require that a paradigm set have at least a certain number of paradigms, unwanted paradigm such as {car-carp, pee-peep} can be ruled out. The Linguistica morphological learner (Goldsmith, 2001) requires a user input parameter for this minimum number of

stems of a given affix pattern.

We further revise the definition of a morphological paradigm:

Definition 11 (morphological paradigms, final version)**.** *A* **morphological paradigm** *is a set of words* $W = \{w_1, w_2, ...\} \subset \Sigma^*$ *such that there exists a stem t, a set of affixes* $A = \{a_1, a_2, ...\} \subset (\Sigma \cup \{\sim\})^*$*, and* $W = t \oplus A$*. Ideally, this stem t is an element of a set of stems* $T = \{t_1, t_2, ..., t_k\}$ *that can compose words with the same affix set A to form a paradigm set with k morphological paradigms.*

2.2.3 Formulating the problem of stem extraction

With all components of a morphological paradigm in place, stem extraction of a morphological paradigm W can be defined as the function of $argmin_{t,A} Cost(t, A)$ such that $W = t \oplus A$. That is, for a given morphological paradigm, stem extraction finds a stem and its associated affixes compatible to this paradigm, such that the cost of the analysis due to the stem and its affixes is at the minimum.

The cost function takes a stem t and a set of affixes A and returns their total cost; a simple version can be the number of letters of the stem and all the affixes.

Taking the argmin follows the MDL principle (section 1.4.3). Given a morphological paradigm, the word forms can potentially be decomposed in multiple ways to yield different analyses of stems and affixes. If we compute the cost associated with each analysis, MDL provides a guiding principle that the best analysis is the one with the minimum cost.

The stem extraction function of $argmin_{t,A} Cost(t, A)$ tells us how to select a winning tuple of (t, A), given multiple candidate tuples, but does not say *how* to do so – this is the subject of the next section.

2.3 Substrings, subsequences, and submultisets

Several ways of how a stem can be extracted from a morphological paradigm are considered. The basic idea follows the linguistic intuition: the stem is the *maximal common material* of all word forms in a paradigm (Spencer, 2012). The question is what exactly this means and how exactly we can extract the stem in an algorithmically explicit way.

It has turned out that it is useful to think of sequences of symbols in terms of linearity and contiguity. For stem extraction, the basic approach is to extract some common subpart of all word forms in a paradigm. The different combinations of properties in terms of linearity and contiguity give rise to three major ways of interest for pinning down a subpart of a word – substrings, subsequences, and submultisets:

(12) Differences between substrings, subsequences, and multisets

	substrings	subsequences	multisets
linearity	✓	✓	✗
contiguity	✓	✗	✗

Definition 12. *A string x is a* **substring** *of y just in case when $y = vxw$ for some $v, w \in \Sigma^*$.*

To illustrate linearity and contiguity of symbols in a substring of a word, take the string $\langle a, b, c, d, e\rangle$ as an example. From this string, $\langle a, b, c\rangle$ is a substring. $\langle a, c\rangle$ is not, because "a" and "c" are not contiguous in $\langle a, b, c, d, e\rangle$. $\langle b, a\rangle$ is not a substring of $\langle a, b, c, d, e\rangle$, because "a" does not come after "b".

The substring approach chooses the *longest common substring* among the word forms in a paradigm. The substring approach to identifying stems works well for the common type of morphology where different parts of a word are strung together by simple concatenation. For instance, given the present indicative paradigm of Spanish *cantar* 'to sing', *canto-cantas-canta-cantamos-cantáis-cantan*, the longest common substring among these six forms is "cant", which corresponds to what a linguist would say is the stem for the *cantar*

paradigm. However, weaknesses of this approach emerge when there is any deviation from simple concatenation. For illustration, we consider the stem-changing Spanish verb *poder* 'can' whose present indicative paradigm is *puedo-puedes-puede-podemos-podéis-pueden*. For this paradigm, the substring approach to stem identification gives three analyses, with "p", "e", "d" as the three longest common substrings:

(13) Stem as the longest common substring for Spanish verb *poder*

	STEM	*puedo*	*puedes*	*puede*	*podemos*	*podéis*	*pueden*
analysis 1	**p**	**p**uedo	**p**uedes	**p**uede	**p**odemos	**p**odeis	**p**ueden
analysis 2	**e**	pu**e**do	pu**e**des pued**e**s	pu**e**de pued**e**	pod**e**mos	pod**e**is	pu**e**den pued**e**n
analysis 3	**d**	pue**d**o	pue**d**es	pue**d**e	po**d**emos	po**d**eis	pue**d**en

At least *both* "p" and "d", the lexically specific information for the PODER paradigm, should be part of the stem, but none of the three substring analyses above give a stem satisfying this. Therefore, the substring approach is suboptimal for morphology deviating from simple concatenation.

Definition 13. *A* **subsequence** x *of a total word* y *is a partial word such that* $count(\sim, x) \geq count(\sim, y)$ *and there exists* $x' = y$ *where* x' *is* x *but with all occurrences of* $\sim$ *filled by elements in* Σ^*.

For subsequences, an example from above is $\langle j, u, m, p, \sim \rangle$ as a subsequence of $\langle j, u, m, p, e, d \rangle$; $\sim$ in $\langle j, u, m, p, \sim \rangle$ can be replaced by $\langle e, d \rangle$ from Σ^* in order to yield $\langle j, u, m, p, e, d \rangle$. Subsequences obey linearity of symbols but does not necessarily maintain contiguity.

The subsequence approach to stem extraction chooses the *longest common subsequence* among word forms in a paradigm. For the Spanish PODER paradigm:

(14) Stem as the longest common subsequence for Spanish verb *poder*

	STEM	*puedo*	*puedes*	*puede*	*podemos*	*podéis*	*pueden*
analysis 1	**p-d**	p ue d o	p ue d es	p ue d e	p o d emos	p o d eis	p ue d en
analysis 2	**p-e**	p ue do	p ue des p ued es	p ue de p ued e	p od emos	p od eis	p ue den p ued en

The longest common subsequences are "p-d" and "p-e"; the subsequence-based stem "p-d" is desirable because both "p" and "d" are included but the problematic "e" is excluded. This Spanish example also illustrates the point that there can be multiple stem analyses for a given paradigm, which can be teased apart by the MDL principle for comparing analyses (section 1.4.3). In the example here, there are no a priori reasons why an unsupervised learning algorithm could judge whether either "p-d" in analysis 1 or "p-e" in analysis 2 is superior over the other. Consider that we have a Spanish dataset with multiple paradigms including *poder*. All else being equal, due to the two analyses above for *poder*, there are at least two possible grammars g_1 and g_2 for the entire dataset. Given that there are other Spanish paradigms with only one unique subsequence analysis for stem extraction (i.e., the non-stem-changing verbs), the MDL principle would prefer the grammar whose specific set of affixes associated with PODER better match those with other paradigms.

Definition 14. *A* **submultiset** x *of a total word* y *for* $|y| = n$ *is the multiset of symbols* $\{x_1, x_2, ..., x_k\}$ *(possibly with repeating symbols) where* $k < n$ *and* $x_i \in y$ $(0 \leq i < k)$.

The submultiset approach to stem extraction disregards both requirements of linearity and contiguity altogether. It treats each word as if it were a bag of symbols. To identify stem material in a paradigm, the submultiset approach chooses the *largest common submultiset* among the word forms in a paradigm. For Spanish PODER, the submultiset approach gives the unordered {p,d,e} as the stem:

(15) Stem as the largest common submultiset for the Spanish verb *poder*

STEM	*puedo*	*puedes*	*puede*	*podemos*	*podéis*	*pueden*
{p,d,e}	**p**u**ed**o	**p**u**ed**es **p**ue**de**s	**p**u**ed**e **p**ue**de**	**p**o**de**mos	**p**o**de**is	**p**u**ed**en **p**ue**de**n

The submultiset approach improves on the substring approach, because both "p" and "d" are in the submultiset-based stem. However, a new problem arises: {p,d,e} cannot tell if "e" is part of the suffix or is stem-internal between "p" and "d". Abandoning the linear ordering in stem extraction appears to be undesirable.

2.3.1 Space complexity

An important difference between these three concepts of substrings, subsequences, and submultisets for stem extraction is in terms of space complexity. We ask how many substrings, subsequences, and submultisets there are for a given string of length n:

(16) Formulae for counting substrings, subsequences, and submultisets of an n-character string

$$\text{Number of substrings} = 1 + 2 + ... + n = \sum_{k=1}^{n} k$$

$$\text{Number of subsequences} = \binom{n}{1} + \binom{n}{2} + ... + \binom{n}{n} = \sum_{k=1}^{n} \binom{n}{k}$$

$$\text{Number of submultisets} = \binom{n}{1}! + \binom{n}{2}! + ... + \binom{n}{n}! = \sum_{k=1}^{n} \binom{n}{k}!$$

The number for submultisets deserves a note. Submultisets do not necessarily obey linearity, but when we render a submultiset as a word, a linear ordering is needed. This is why we need the factorial of each term of the n-choose-k combinations.

More concretely, let us consider a string of length n for $1 \leq n \leq 10$ in the following table. For instance, a string with ten symbols has 55 possible substrings.

(17) Number of substrings, subsequences, and submultisets for strings of different lengths

n	substrings	subsequences	submultisets
1	1	1	1
2	3	3	3
3	6	7	13
4	10	15	769
5	15	31	7.26E+06
6	21	63	2.43E+18
7	28	127	2.07E+40
8	36	255	1.20E+100
9	45	511	4.74E+211
10	55	1023	2.04E+497

For stem extraction, the worst-case scenario would be that one could exhaustively find all subparts (any of these three approaches) of all the word forms for a given paradigm and the stem will be the longest one that all word forms share. For space complexity, the substring approach is $\mathcal{O}(n \log n)$, the subsequence approach is $\mathcal{O}(n^2)$, and the submultiset approach is $\mathcal{O}(n!)$.

Fortunately, in practice, regardless of whether we pick the substring, subsequence, or submultiset approach, we simply have to consider the subpart options of the *shortest* word for a given paradigm. Under our assumption of the stem being the maximal common material for all word forms, the paradigm jump-jumped-jumping-jumps cannot possibly have a stem longer than "jump".

Moreover, linguistically it is clear that linearity is important in natural languages, and that the submultiset approach (the worst approach for space complexity) is untenable.

Among the three approaches considered, the subsequence approach for stem extraction appears the most promising, both for being able to handle concatenative and non-concatenative morphology, as well as for not having a space complexity as high as that of the submultiset approach.

2.4 Results

This section illustrates the results of stem extraction by the methods of longest common substring, longest common subsequence, and largest common submultiset, using morphological paradigms from English and Arabic. Detailed outputs for English are provided in the appendix.

It will become clear that, if we are in search of a language-independent stem extraction method, it is reasonable to model the stem as the longest common subsequence in a morphological paradigm, among the options explored in this work. The substring approach is too restrictive, as it breaks down as soon as there is any kind of non-concatenative morphology. The submultiset approach accommodates non-concatenative stems, but at the cost of admitting a large amount of possible affixes.

The cost function takes the number of symbols—letters and the $\sim$ wild card —in a given analysis (stems plus affixes for each stem) and multiplies this number by five. The factor of five is to get an approximate cost in terms of bits, as $2^5 = 32$, roughly the number of symbols in the Latin alphabet.

2.4.1 English

The English data is verbal morphological paradigms from the top 100 most frequent verbs according to COCA. Here are the top 10 in order: be, have, do, say, go, get, know, make, think, take. For this dataset, my implementation as well as the cost function, modeling the stem as the longest common subsequence incurs the lowest cost of 11,619, compared to

12,005 for the substring approach and 14,329 for the submultiset approach.

Pure concatenative morphology does not distinguish between the substring and subsequence approaches by cost. This is because a substring can be viewed as a special case of a subsequence, where all symbols happen to be contiguous. The following is the result of the WANT paradigm. Both substring and subsequence analyses each cost 85, which is the symbol count of { want, ∼, ∼s, ∼ed, ∼ed, ∼ing } (17 symbols) times 5.

(18) Stem extraction for English verb WANT

want, wants, wanted, wanted, wanting

substring (cost: 85)

want	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

want	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 116)

antw	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing, ␣␣n␣i␣g
	∼	∼s	∼ed	∼ed	∼ing, ∼n∼i∼g

The submultiset analysis for WANT exposes the problem that abandoning linearity would lead to multiple analyses in the affixes that would increase the cost of the analysis. In the word form "wanting", the stem is the multiset $\{a, n, t, w\}$, whereas the corresponding affix is an indeterminacy between $\langle ␣, ␣, ␣, ␣, i, n, g\rangle$ and $\langle ␣, ␣, n, ␣, i, ␣, g\rangle$ due to the ambiguous "n".

When there is at least one suppletive form in the paradigm, all the three stem extraction methods output an empty stem, and therefore all the affixes are identical to the word forms. The BE paradigm is an example:

(19) Stem extraction for English verb BE

be, is, was, been, being

substring (cost: 80)

subsequence (cost: 80)

submultiset (cost: 80)

The GET paradigm is an example with non-concatenative morphology that shows the subsequence approach for stem extraction is superior to the substring or submultiset approach.

(20) Stem extraction for English verb GET

get, gets, got, gotten, getting

substring (cost: 344)

g	␣et	␣ets	␣ot	␣otten	␣etting, gettin␣
	∼et	∼ets	∼ot	∼otten	∼etting, gettin∼
t	ge␣	ge␣s	go␣	go␣ten, got␣en	ge␣ting, get␣ing
	ge∼	ge∼s	go∼	go∼ten, got∼en	ge∼ting, get∼ing

subsequence (cost: 192)

gt	␣e␣	␣e␣s	␣o␣	␣o␣ten, ␣ot␣en	␣e␣ting, ␣et␣ing
	∼e∼	∼e∼s	∼o∼	∼o∼ten, ∼ot∼en	∼e∼ting, ∼et∼ing

submultiset (cost: 263)

gt	␣e␣	␣e␣s	␣o␣	␣o␣ten, ␣ot␣en	␣e␣ting, ␣et␣ing, ge␣tin␣, get␣in␣
	∼e∼	∼e∼s	∼o∼	∼o∼ten, ∼ot∼en	∼e∼ting, ∼et∼ing, ge∼tin∼, get∼in∼

Apart from GET, other paradigms listed in the English results in the Appendix with similar non-concatenative morphology include TAKE, COME, FIND, COME, TELL, GIVE.

2.4.2 Arabic

The Arabic data is 20 verbal paradigms. Similar to the English results, the subsequence approach outperforms the other two for most verbs. Overall, in our settings, the subsequence approach incurs a cost of 24,038, compared to 29,510 for submultisets and 48,018 for substrings.

The root-and-pattern morphology highlights the sharp contrast between the substring and subsequence approaches to stem extraction. Consider the "write" paradigm with k-t-b:

katabtu

katabtu, katabta, katabti, kataba, katabat, katabnaa, katabtum, katabtunna, katabuu, katabna, aktubu, taktubu, taktubiyna, yaktubu, taktubu, naktubu, taktubuuna, taktubna, yaktubuuna, yaktubna

substring (cost: 4439)

a	k␣tabtu, kat␣btu	k␣tabta, kat␣bta, katabt␣	k␣tabti, kat␣bti	k␣taba, kat␣ba, katab␣
b	kata␣tu	kata␣ta	kata␣ti	kata␣a
k	␣atabtu	␣atabta	␣atabti	␣ataba
t	ka␣abtu, katab␣u	ka␣abta, katab␣a	ka␣abti, katab␣i	ka␣aba

subsequence (cost: 1471)

atb	k␣␣a␣tu	k␣␣a␣ta	k␣␣a␣ti	k␣␣a␣a	k␣␣a␣at	k␣␣a␣naa	k␣␣a␣tum	k␣␣a␣tunna	k␣␣a␣uu
	k∼a∼tu	k∼a∼ta	k∼a∼ti	k∼a∼a	k∼a∼at	k∼a∼naa	k∼a∼tum	k∼a∼tunna	k∼a∼uu
ktb	␣a␣a␣tu	␣a␣a␣ta	␣a␣a␣ti	␣a␣a␣a	␣a␣a␣at	␣a␣a␣naa	␣a␣a␣tum	␣a␣a␣tunna	␣a␣a␣uu
	∼a∼a∼tu	∼a∼a∼ta	∼a∼a∼ti	∼a∼a∼a	∼a∼a∼at	∼a∼a∼naa	∼a∼a∼tum	∼a∼a∼tunna	∼a∼a∼uu

submultiset (cost: 1824)

abkt	␣␣␣a␣tu, ␣␣ta␣␣u, ␣a␣␣␣tu, ␣at␣␣␣u	␣␣␣a␣ta, ␣␣ta␣␣a, ␣a␣␣␣ta, ␣a␣a␣t␣, ␣at␣␣␣a, ␣ata␣␣␣
	∼a∼tu, ∼at∼u, ∼ta∼u	∼a∼a∼t∼, ∼a∼ta, ∼at∼a, ∼ata∼, ∼ta∼a

In the subsequence analyses, k-t-b, as would be desired in a linguistic analysis, is one of the extracted stem. In contrast, the substring analyses are forced to list each of the symbols k-t-b in the individual extracted stems, thereby increasing the overall cost of the substring approach by a fair amount. Similar to English "wanting" above, the submultiset analyses bear additional costs due to ambiguous symbols, particularly the intervening vowels for Arabic here.

2.5 Remarks

2.5.1 Stem allomorphy

The example with Spanish PODER above illustrates a phenomenon common across the world's languages: stem allomorphy. From the perspective of Spanish morphophonology, a standard analysis treats the vowel-alternating patterns such as *o*∼*ue* in PODER as diphthongization in connection with stress shifting. Under this view, it is legitimate to say that *pod-* and *pued-* are stem allomorphs for PODER in present indicative. However, stem identification in the context of this dissertation makes a binary distinction of stem versus affixal material within a paradigm, where some phonological material being shared by all word forms in a paradigm is the necessary and sufficient condition to qualify as stem material. This is why only "p-d" for PODER above is regarded as the stem by the subsequence analysis, and everything else, including the *o*∼*ue* alternation together with the inflectional suffixes, is considered affixal material.

At first blush, this might appear counter-intuitive, but the goal of this dissertation research is to develop language-independent strategies of morphological analysis (no knowledge of diachronic development, meaning, and so forth). In the case of Spanish verbs, such philosophy of linguistic analysis is actually beneficial. Spanish verbs are well known to consist of both stem-changing (like PODER with *o*∼*ue*) and non-stem-changing stems. This means that

when we examine structure *across* morphological paradigms in Spanish verbs, such a distinction of allomorphy (or the lack thereof) should emerge in one way or another. Treating *o∼ue* and the like as affix material leads to the straightforward computational treatment that all material not shared by all word forms in a paradigm will be subject to cross-paradigmatic comparison.

2.5.2 Linearity, contiguity, and morphemes

Stem identification is an essential step in morphological analysis. Its importance has long been recognized (Nida, 1949). My doctoral research with respect to stem identification invites rethinking of assumptions that have been implicitly taken for granted: linearity versus contiguity and the nature of morphemes.

The alphabetic writing system is used extensively both within and without linguistic research. Its properties are therefore carried over throughout linguistic analyses. Of particular interest here is the assumption of linearity: we represent language using a finite set of symbols concatenated linearly. There are two important properties: (i) there is one and only one tier, to use an autosegmental term; and (ii) contiguity of adjacent symbols is assumed. The assumption of linearity and contiguity has been explicitly discussed in connection with the strictly linear use of phonemes to represent language (de Saussure, 1916; Hockett, 1947). While strict linearity was still assumed in classical generative phonology (Chomsky and Halle, 1968), various linguistic phenomena for which a *multilinear* representational system is needed have been identified, e.g., Goldsmith (1976, 1990) on tone and McCarthy (1979) on root-and-pattern morphology; Ladd (2014) also critically examines the notion of linearity in phonology. At first blush—and rightly so—Goldsmith's work differs from McCarthy's: Goldsmith's work is in part motivated by the simultaneity of tone and consonant-vowel sequencing, while McCarthy's deals with the non-contiguity across roots and vocalic patterns in Semitic languages. Importantly, their work shares the property that

the segments (tone, consonant, or vowel) in each tier of a multilinear representation are linearly ordered. My doctoral research on stem identification also points to the conclusion that the linear ordering among segments is essential; the submultiset approach is suboptimal. However, contiguity—though misleadingly and inevitably encoded in the alphabetic writing system—does not necessarily hold in stem identification, very much like what McCarthy's autosegmental treatment of Semitic languages attempts to demonstrate. Discarding contiguity while retaining linear ordering appears to be a promising approach to stem identification, especially for non-concatenative morphology ranging across root-and-pattern morphology, stem allomorphy, and infixation (Yu, 2007). Research in phonology also supports this view, especially recent works on how non-local phonological processes (e.g., harmony phenomena) can be modeled as local processes (Heinz et al., 2011; Goldsmith and Riggle, 2012).

The departure from contiguity inherent in the alphabetic writing system begs the question of how we are to understand the concept of morphemes. The various stem identification algorithms described in this chapter make the explicit assumption that a word is composed of some stem material plus some affixal material and nothing else, but there is no formal commitment to the concept of morphemes. Because the principal goal of my dissertation research is to devise computational tools *without* language-specific knowledge, two consequences follow. First, the datasets (paradigm tables for supervised learning or unlabeled raw text for unsupervised learning) do not encode meaning. Second, as a corollary of the first consequence, what we call "affix" is entirely analogous with non-stem material, and there is no attempt to further segment affixes into smaller pieces (except when the stem is non-contiguous and therefore the affix is intertwined with the stem). Without meaning from the input data, we are not concerned with issues of form-meaning mapping. The stems and affixes, understood purely in terms of their surface forms in this dissertation, might be called morphemes for convenience, but morphemes do not have a formal status in this dissertation research. This brings us closer to theories of morphology that reject the notion of morphemes

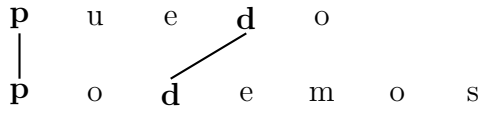
(Anderson, 1992) and approaches to morphological analysis without morpheme consistency, e.g., Pham and Lee (2014, 2018) on truncation.

2.5.3 Correspondence

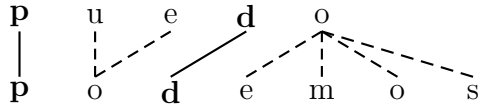
Extracting the common material as the stem from a morphological paradigm and treating the residue as the affix in each word form makes all the word forms stand in an alignment relationship. Take *puedo∼podemos* from the Spanish discussion above as an example:

(21) Alignment between *puedo∼podemos*

a. stem alignment



b. affix alignment



Stem identification is closely related to a wide variety of linguistic phenomena. The longest common subsequence approach for the Spanish PODER present indicative paradigm, as described in this section, identifies p-d as the stem. In other words, all the six word forms in the PODER paradigm have p-d aligned as the same element. With the pair *puedo∼podemos* for a simplified illustration, this stem alignment is visualized in (21a). What is not aligned is the affix in each word form. If all affixal elements across word forms are also aligned such that there is no line crossing and all elements are aligned (very much reminiscent of the Well-formedness Conditions in Goldsmith's autosegmental phonology), then affix alignment is illustrated for the current example with dashed lines in (21b). Such alignment relationships among word forms are among the basis of research on the interaction between morphology

and phonology, most notably the large body of work that stems from the Correspondence Theory (McCarthy and Prince, 1995), proposed largely in the context of modeling reduplication in Optimality Theory. Phenomena that hinge on some correspondence relationship among morphologically related words (reduplication, paradigmatic effects such as apparent uniformity, cyclic effects, and so forth; Steriade (2009); Inkelas (2014)) have been studied mostly in a way where the exact correspondence relationships of the segments appear to come from the analyst's intuition rather than an explicit algorithm for *how* correspondence is computed. Algorithmic approaches to stem identification will shed light on the methodological, and ultimately theoretical, issues involved.

CHAPTER 3

PARADIGM SIMILARITY

3.1 Introduction

While the previous chapter is about structure *within* a morphological paradigm, this chapter is about structure *across* morphological paradigms. We are interested in learning structure from morphological paradigmatic data such as (22) for English verbs.

(22)

jump	jumping	jumps	jumper	jumped
walked	walks	walker	walking	walk
moving	mover	moved	moves	move
loves	love	loving	mover	loved
⋮	⋮	⋮	⋮	⋮

Data such as (22) has several characteristics that are central to the learning task of interest. Each row represents one paradigm from a lexeme (JUMP, WALK, MOVE, etc.; small caps denote lexemes in the linguistic convention). All rows have the same number of forms for distinct morphological realizations; there are five forms in each row in (22). Moreover, all rows have forms for the exact same morphosyntactic categories. In (22), it is always the same five morphosyntactic categories of English verbal paradigms in every row: the bare form, the third singular present sense with *-s*, the simple past typically with *-ed*, and the *-ing* form. Lastly, within a row, the different forms of the paradigm can be horizontally ordered in an arbitrary way. (22) happens to be English verbs, but it could have been from another language, or from another part of speech.

Given a paradigm data set like (22), we ask if we are able to learn structure algorithmically. Specifically, in this chapter we are interested in clustering. We propose an algorithm which learns the cross-paradigmatic structure based purely on surface strings for (i) morphological groupings of the paradigms akin to conjugation and declension classes, and (ii)

the hierarchical patterns among these morphological groupings.

This chapter is based on results reported in Lee (2014).

3.2 Clustering morphological paradigms

In many languages with inflectional morphology, paradigms tend to exhibit patterns which form groups, or *clusters*. For verbal paradigms, we call them conjugation classes. For paradigms of other parts of speech, we call them declension classes. In this chapter, the goal of clustering is to algorithmically find out these conjugation/declension classes given some paradigmatic data.

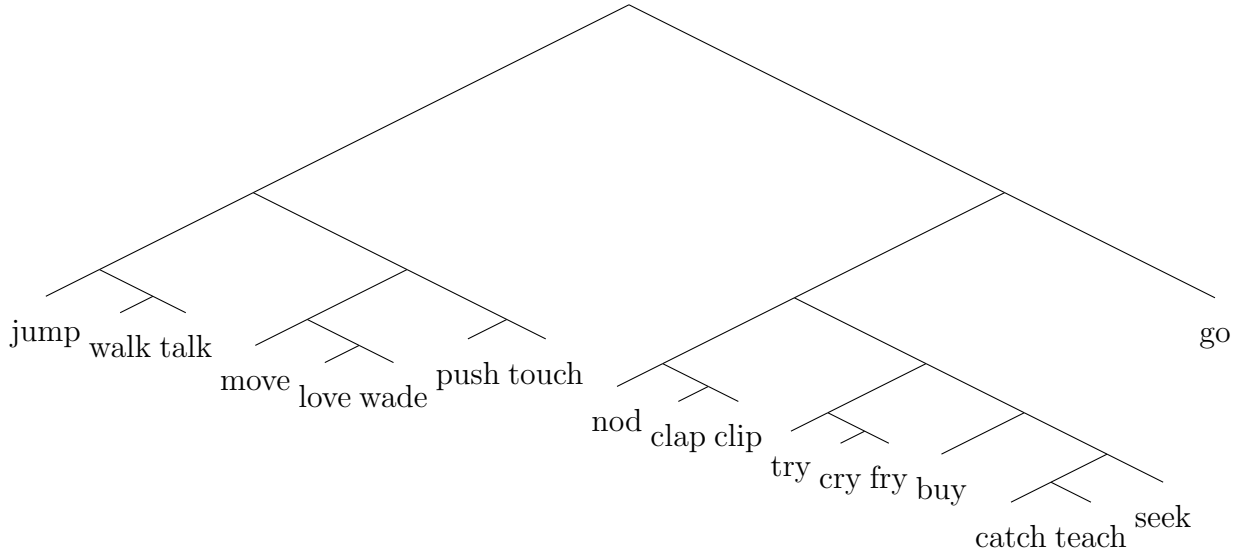
Arguably, English verbal paradigms in (22) also display conjugation classes (Bloch, 1947). For instance, there is a group of verbs which share the ablaut pattern in *sing-sang-sung, drink-drank-drunk*. If we allow a broader reading of 'conjugation' for English, then we may admit orthographical alternations as well, at least in the sense that how we write present-day English indeed reflects a non-contemporary version of English, or that the orthography shows what we observe in other languages with well-established inflection classes. For (22) plus a few more paradigms, some 'conjugation classes' for English are as follows:

(23) Some English 'conjugation classes'

Property	**Paradigms**				
Regular	jump	jumping	jumps	jumped	jumper
	walk	walking	walks	walked	walker
With 'e'	move	moving	moves	moved	mover
	love	loving	loves	loved	lover
Consonant doubling	nod	nodding	nods	nodded	nodder
	clap	clapping	claps	clapped	clapper
'y/ie'	try	trying	tries	tried	tryer
	cry	crying	cries	cried	cryer
$\{o,a\}ught$	buy	buying	buys	bought	buyer
	catch	catching	catches	caught	catcher

Our task is to perform clustering on morphological paradigms like these English verbs, based purely on surface strings. As a quick illustration whose details are to be discussed in subsequent sections, our algorithm produces the following hierarchical representation (very much simplified here) for an English data set similar to (22).

(24) An inheritance hierarchy for some English 'conjugation classes'



Not only do we attempt to look for the inflection classes, we also infer higher-order structure, namely the hierarchical relationship among the inflection classes. This is linguistically significant, because many inflectional morphological systems do not have entirely distinct string-based patterns across inflection classes, although the number of classes could logically be as many as there are lexemes; this observation is termed *paradigm economy* in Carstairs (1983, 1987). The different combinations of some small number of inflectional patterns are what result in the partial overlapping and similarity among inflection classes.

3.3 On inflection classes

This section explains why linguists should care about clustering introduced in the previous section. The linguistic relevance of clustering is *inflection classes*, a well-known yet understudied linguistic phenomenon. If inflectional morphology is what is relevant to syntax (Anderson, 1982), then the very existence of inflection classes presents challenges to our understanding of language. Inflection classes are the groupings, usually but not always arbitrary, of the lexemes of a given lexical category in terms of inflectional patterns; their existence is often attributed to diachronic reasons, see Dammel (2009) for a case study. A familiar example of inflection classes is Spanish verbal morphology, which has three major groups— -AR, -ER, and -IR verbs— with no phonological, syntactic, or semantic basis. In the call for papers for the special session on "Inflectional Classes in the Languages of the Americas" at the 87$^{\text{th}}$ Annual Meeting of the Linguistic Society of America in 2013, the crux of the problem and the reason why we are interested in inflection classes are aptly summarized:

> "Inflection classes are seemingly useless in functional terms, and yet they are widely found across languages and remarkably resilient over time. [...] Inflection classes, as they resist a syntactic or phonological explanation, are in themselves an interesting object of study for a theory of language because they introduce

into the linguistic system a layer of complexity which is purely morphological."[1]

A quick survey of morphology textbooks and linguistics glossaries reveals that the treatment of and attention to inflection classes are somewhat uneven (Jensen, 1990; Spencer, 1991; Trask, 1999; Haspelmath, 2002; Bauer, 2003, 2004; Matthews, 2007; Crystal, 2008; Haspelmath and Sims, 2010; Lieber, 2010; Aronoff and Fudeman, 2011). In more technical works, the situation is similar. In formal studies of inflectional morphology, so long as inflection classes are not the focus of discussion, a popular treatment of inflection classes is simply assign diacritic or class features of some sort to lexemes. For example, a binary feature such as [±strong] is used to distinguish strong and weak verbs in English in Distributed Morphology (Halle and Marantz, 1993).

3.3.1 The connection between inflection classes and clustering

This section underscores the connection between inflection classes and clustering. The discussion will be illustrated with the well-known Spanish conjugation classes:

(25) The Spanish present indicative suffixes

	1st conjugation	2nd conjugation	3rd conjugation
1.SG	-o	-o	-o
2.SG	-as	-es	-es
3.SG	-a	-e	-e
1.PL	-amos	-emos	-imos
2.PL	-áis	-éis	-ís
3.PL	-an	-en	-en

Clustering of morphological paradigms is tantamount to learning inflection classes and their string-based hierarchical relationship. If descriptive grammars and current morpholog-

1. From `http://linguistlist.org/callconf/browse-conf-action.cfm?ConfID=147727`

ical theory recognize inflection classes, or at least take inflectional morphology seriously (cf. Aronoff 1994; Blevins 2006; Matthews 1972; Spencer 1991; Stump 2001a, and others), one must ask whether (and how) these classes can be learned. Once the inflection classes are established, we are also curious if there exists any structure *across* them. Cross-linguistically, it is observed that inflection classes exhibit partial similarity – part of Carstairs' (1983; 1987) notion of paradigm economy. The Spanish conjugation system as presented in (25) provides a convenient illustration: intuitively, -ER and -IR verbs are more similar to each other than either to -AR verbs.

As the table in (25) above shows, the three Spanish conjugation classes are distinct for the first- and second-person plurals, e.g., *-amos, -emos*, and *-imos* for the first-person plurals. It is differences of this type which give rise to inflection classes. At the same time, these conjugation classes share a great deal in common. Across all three classes, the first-person singular suffixes are *-o*, the second-person singular suffixes end with *-s*, and so forth. It is these similarities which make alignment possible. An algorithmic approach to alignment and clustering will reveal structured similarities and differences across morphological paradigms.

3.3.2 String-based inheritance hierarchies

Inflection classes do not differ from one another in an arbitrary way. There is a good amount of partial similarity among inflection classes (Matthews, 1991), and there appears to be an upper bound of the number of inflection classes given the number of inflectional affixes (Müller, 2007). As such similarity is usually uneven across inflection classes in various combinations, it is reasonable to think of inflection classes as having a nested structure (Corbett and Fraser, 1993; Stump, 2001b) based on the complex overlapping patterns; these patterns have been studied in depth in terms of principal parts (Stump and Finkel, 2013), specific frameworks such as Network Morphology (Brown and Hippisley, 2012), and a combination of these (Baerman, 2012). In general, to study structure of this type, clustering is a suitable

and explicit tool to explore and display the hierarchical structure of inflection classes. In the following, we discuss a highly intuitive and illustrative example on Greek nominals by Haspelmath (2002). We will see that some of Haspelmath's goals and results are very similar to ours in this chapter.

Haspelmath considers seven declension classes of Greek nominals, in (26), and discusses what he calls the inheritance hierarchies among them. The very fact that his data have the morphosyntactic features such as number, gender, and case means that we are not dealing with alignment. Working out the inheritance hierarchies among inflection classes for Haspelmath is exactly our clustering problem.

(26) Seven classes of Greek nominals, data from Haspelmath (2002, 125)

	SG			PL			
Class	NOM	ACC	GEN	NOM	ACC	GEN	
os	nomos	nomo	nomu	nomi	nomus	nomon	'law (masc.)'
as	pateras	patera	patera	pateres	pateres	pateron	'father (masc.)'
us	papus	papu	papu	papuðes	papuðes	papuðon	'grandfather (masc.)
a	imera	imera	imeras	imeres	imeres	imeron	'day (fem.)'
i1	texni	texni	texnis	texnes	texnes	texnon	'art, skill (fem.)'
i2	poli	poli	poleos	poles	poles	poleon	'town (fem.)'
u	maimu	maimu	maimus	maimuðes	maimuðes	maimuðon	'monkey (fem.)'

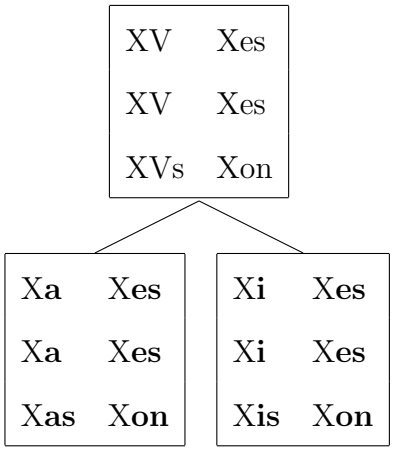
Haspelmath observes that, in terms of surface forms, these declension classes are not completely distinct from one another. The partial similarity is illustrated by comparing the *a*- and *i1*-declension classes.

(27) Greek *a*- and *i1*-declension classes, from Haspelmath (2002, 125)

	a-declension	*i1*-declension
SG NOM	imer**a**	texn**i**
ACC	imer**a**	texn**i**
GEN	imer**as**	texn**is**
PL NOM	imer**es**	texn**es**
ACC	imer**es**	texn**es**
GEN	imer**on**	texn**on**
	'day (fem.)'	'art, skill (fem.)

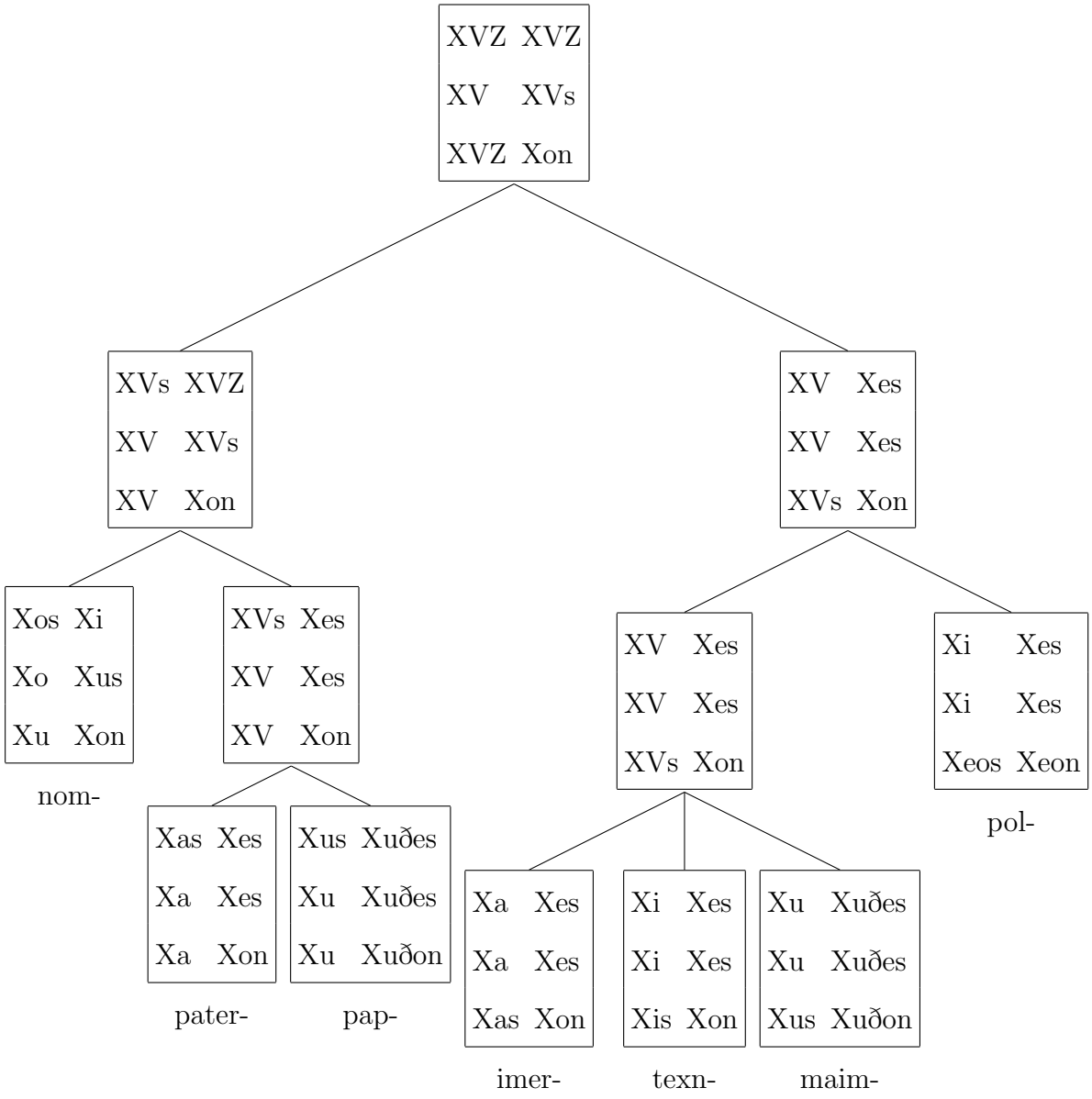
If we replace the common portion among all the forms within a class by "X", it becomes clear that these two classes differ by only a vowel in the singular forms. Haspelmath illustrates this by means of "V" as a variable for the vowel, and postulates a common template for both classes in a hierarchical representation:

(28) The hierarchy of *a*- and *i1*-declension classes, from Haspelmath (2002, 127)



Then, Haspelmath makes a giant leap forward, by presenting an inheritance hierarchical analysis of all the seven Greek nominal classes we saw earlier.

(29) An inheritance hierarchy of seven Greek declension classes, from Haspelmath (2002, 128)



The tree in (29) can be a sophisticated answer to the question of how many classes there are among the seven Greek paradigms concerned. It may range from two (*macro-classes* in Haspelmath's terminology), referring to masculine classes (the three leaves together on the left in (29)) and feminine ones (the other four leaves on the right), to four or seven (*micro-classes*). Furthermore, the tree also indicates the defaults and principal parts for the given data. The plural genitive can be considered a default, since all the given seven

paradigms end with *-on*. The top node with "Xon" for plural genitive in (29) indicates that all paradigms in question here share, or *inherit*, such a morphological characteristic. On the other hand, the singular genitive is a possible principal part, because the seven paradigms have almost distinct realizations (with six unique ones) for this cell: *-u, -a, -u, -as, -is, -us, -eos*.

Both Haspelmath's work and ours are about learning inflection classes and their hierarchical patterning, but we do not assume knowledge of morphosyntactic alignment at the outset. Interestingly, Haspelmath does not explain how exactly the rest of the tree with hierarchical clusters in (29) is obtained. We work on the same clustering problem, with fewer assumptions, and more explicitly.

3.4 Algorithm

This section describes in detail the algorithm to perform alignment and clustering. In brief, we treat the tasks at hand as an iterative, greedy optimization problem. At each iteration, the *complexity* of the system, to be defined and explained below, is minimized. Here is the algorithm in brief pseudocode:

(30) The alignment-clustering algorithm

```
   Data: n paradigms, each with k forms
 1 Initialize stems and affixes;
 2     n paradigms → n stemplexes;
 3 Initialize overall complexity (grammar cost + data cost);
 4 while n > 1 do
 5     for i ← 1 to (n 2) do
 6         (for the i-th merging possibility of the 2 stemplexes);
 7         for j ← 1 to k! do
 8             Compute d_ij, the decrease in complexity for the j-th alignment choice of
                the i-th merging possibility;
 9     For the largest d_ij, actually perform the i-th merging for those 2 stemplexes at
        the j-th alignment choice;
10     n ⟵ n − 1;
```

3.4.1 Initializing stemplexes

The first initialization step is to create stemplexes from the paradigms. We have felt the need to coin the word *stemplex* which refers to a new entity: a composite object with a list of stems, a list of (union) affixes, and a list of morphological paradigms which look very much like the input data. In its simplest form, a stemplex has only one morphological paradigm, together with its stem and affixes.

The starting point is a data set with n rows (paradigms), each with k forms:

(31) An $n \times k$ data set

	k **columns**				
n **rows**	jump	jumping	jumps	jumped	jumper
	clap	clapping	claps	clapped	clapper
	⋮	⋮	⋮	⋮	⋮

Stems and affixes

For each paradigm, the algorithm defines its stem and affixes. A stem is the multiset of the letters/sounds shared by all the forms in the paradigm, and what remains in each form is an affix; the detailed discussion on strings as multisets is in section 3.4.1. Deriving stems and affixes is not the focus of this paper, but we need some hypothesis to get off the ground (as in Haspelmath's work on Greek nominals discussed above) for alignment and clustering. As it stands, the algorithm does not update the stems and affixes once they are initialized. Some paradigms with their stem and affixes are illustrated below.

(32) Deriving stems and affixes by the algorithm

Paradigm	Stem + affixes
jump, jumping, jumps, jumped	jump + {Ø, ing, s, ed}
clap, clapping, claps, clapped	clap + {Ø, ping, s, ped}
go, going, goes, went	Ø + {go, going, goes, went}

If one examines the stems and affixes in (32, it appears at first blush that some of them do not quite make sense to a linguist. In particular, the GO paradigm has the null string Ø as its stem, and all the forms of this paradigm have been shoveled to the affix slots. There is no bug here: this is necessarily the case because the verb forms *go* and *went* do not share any letters at all. Indeed, the stem and affixes thus derived for GO may look odd, but as alluded to above, getting the stems and affixes right (however one defines "right" here) is tangential to alignment and clustering, our main goals in this chapter.[2] To use the airplane-versus-bird analogy as in Jurafsky and Martin (2006, 14), both airplanes and birds have wings, both fly, but airplanes do not flap their wings. In our case, we think that we do need some hypothesis of stems and affixes, as many linguists do when they work on morphology.[3] However, we

2. Goldsmith (2011b) proposes a string algebra to learn morphophonology in paradigms, which has the potential to learn stem allomorphy between Ø and *go* for GO, for instance.

3. This is naturally true for a linguist operating within a morpheme-based framework of morphology and

need stems and affixes for other purposes, and therefore having our stems and affixes match what one would think they should look like is a non-issue. As will be clear in section 3.5 on results, a paradigm such as GO with a peculiar stem-affix hypothesis does not have adverse effects in alignment and clustering. Naturally, there have been explicit attempts to infer the stem and affixes of a morphological paradigm with varying assumptions (see the survey on morpheme segmentation in Goldsmith 2010; Hammarström and Borin 2011), but this is not our objective here.

Strings as multisets of symbols

An important remark is in order regarding the representation of strings. Our algorithm actually treats all word forms, stems, and affixes as *bags of letters* with both adjacency and linear ordering of letters/sounds removed.[4] For example, the word *jump* is computationally represented as the alphabetized "jmpu", if it has to be represented as a string at all. This strategy has multiple advantages. First, it makes it computationally easy to derive stems and affixes. Second, it does not assume whether the language at hand is a prefixing, suffixing, or even infixing language.[5] This point is illustrated by the potential of the algorithm to deal with languages with templatic morphology with consonantal roots such as Semitic languages.[6] To give a concrete example, we use the well-known Arabic forms with the triconsonantal root *k-t-b* (loosely meaning 'to write'):

---

morphosyntax. But even in word-based approaches, it is difficult, if not impossible, to get away from the idea that a morphologically complex word typically has some phonological material (i.e., the affixal exponence) shared by other words (Anderson, 1992; Booij, 2010).

4. It is also possible to treats strings with only linear ordering while adjacency is ignored. For stem extraction, this way of treating word forms leads to the idea that the stem is the longest common *subsequence* of all word forms in a paradigm, cf. the previous chapter.

5. Austronesian languages, for example, are well-known to employ infixation in their inflectional morphology; see Yu (2007) for a general survey of infixation.

6. For languages with templatic morphology, there are sophisticated unsupervised approaches such as Goldsmith and Xanthos (2009), who use a graph theoretical approach to separate vowels and consonants in order to learn consonantal roots in Arabic.

(33) Arabic *k-t-b* forms and their alphabetized representations in the algorithm [7]

	Arabic form	Alphabetized representation
'he wrote'	**k**a**t**a**b**a	aaa**bkt**
'we wrote'	**k**a**t**a**b**nā	aaā**bk**n**t**
'he writes, will write'	ya**kt**u**b**u	a**bkt**uuy
'we write, will write'	na**kt**u**b**u	a**bk**n**t**uu
'writer'	**k**ā**t**i**b**	ā**bikt**
'he dictated'	a**kt**a**b**a	aaa**bkt**
'he dictates, will dictate'	yu**kt**i**b**u	**bikt**uuy
'he asked s.o. to write s.th.'	ista**kt**a**b**a	aaa**bikst**t
(imperfect of above)	yasta**kt**i**b**u	aa**bikst**tuy
'office'	ma**kt**a**b**	aa**bk**m**t**

Using these Arabic forms for illustration, the way the algorithm derives the stem and affixes for a given paradigm is as follows. First, the shortest word form is located, i.e., *ābikt* (alphabetized for *kātib*). Then, the algorithm scans it from left to right, asking whether each letter is present in all word forms. If it is, the letter is part of the stem, but if not, the letter is part of an affix. In this case, for *ābikt*, only *bkt* is shared by all word forms in (33), and therefore is the stem. This stem is exactly the alphabetized version of the triconsonantal root *k-t-b*. Right from Harris (1955) on the unsupervised learning of morphological structure, it is customary that the linear ordering of letters or sounds is assumed and used. Indeed, this necessarily has to be the case for tasks such as morpheme segmentation. But for objectives like ours, if discovering morphemes is at best secondary, then removing the linear ordering of letters gives rise to interesting and useful consequences.

Using our terminology, we have n stemplexes initialized from n paradigms. Next, the

7. Arabic forms are from `http://en.wikipedia.org/wiki/Semitic_root` accessed May 16, 2013. The way in which the orthographic *ā* (for /aː/) is alphabetized does not affect our points of interest.

algorithm initializes an important measurement, the complexity of the stemplexes.

3.4.2 The complexity computation

What is the complexity of an analysis? The notion of complexity is frequently appealed to in theoretical linguistics. A typical scenario consists of multiple competing analyses from different frameworks for some given linguistic data, and the argumentation goes in favor of one analysis, argued to be the least complex, and therefore the particular theoretical framework within which the analysis is couched is superior. Complexity is often characterized qualitatively without rigorous quantification. This makes comparison of analyses rest on intuitive and subjective terms. This paper represents a step forward towards an objective and testable measure of complexity. For our purposes, we speak of the complexity computation of some morphological analysis. Two distinct but related questions are asked: (i) How is complexity computed? (ii) How is complexity used?

This section focuses on providing an answer to the first question. We propose a way to represent the complexity of a stemplex, using a number whose computation is detailed below. The next section is to answer the second question, where we detail the use of the complexity measurements in the algorithmic steps of alignment and clustering.

The complexity computation is explained by means of an example. Consider the JUMP stemplex with five word forms:[8]

8. As explained above, the algorithm actually treats everything as bags of letters. For reasons of readability, however, we use the human-friendly representations throughout this paper.

(34) The JUMP stemplex

		jump	jumps	jumping	jumped	jumper	⇐ TARGET FORMS
STEM ⇒	jump	Ø	s	ing	ed	er	⇐ AFFIXES

Complexity is a trade-off between two related components:

(35) Complexity = Grammar cost + Data cost given the grammar

Some remarks are in order as to why we need both components. First, the grammar cost measures how complex a grammar is. Second, we also need a measure for the forms actually observed, and that is the data cost. We need both costs because a grammar provides only generalizations in abstract terms for some given data but not the actual linguistic forms we use. For instance, we may well utter the sentence *The dog chased the cat* but not *Article Noun Verb Article Noun* given by a part-of-speech analysis. The actual forms are generated by the grammar, and they incur a cost.

The grammar cost is dependent on the set of p stems $\{t_1, \ldots, t_p\}$ and affixes $\{x_1, \ldots, x_k\}$:

(36) Grammar cost $= \lambda \cdot (\sum_{i=1}^{p} |t_i| + \sum_{j=1}^{k} |x_j| + k)$

The grammar cost of a stemplex composed of p paradigms hinges on three terms: the length of all the p stems ($|s|$ denotes the length of a string s), the length of all the k affixes, and k (the number of word forms in each paradigm). λ is set to be 5, because $2^5 = 32$ is roughly the number of letters in the alphabet for languages of interest, and five bits are needed to encode one letter. To illustrate the computation, we use the JUMP stemplex as in (34), i.e., $p = 1$, with only one paradigm in this stemplex, and $k = 5$ for five word forms.

(37) Grammar cost of the JUMP stemplex[9]

9. As it stands currently, the algorithm treats the null string Ø as a letter of zero letters long.

$$
\begin{aligned}
& 5 \cdot (|jump| + |\emptyset| + |s| + |ing| + |ed| + |er| + 5) \\
=& 5 \cdot (4 + 0 + 1 + 3 + 2 + 2 + 5) \\
=& 85
\end{aligned}
$$

The data cost computation is arguably more complicated. Several parameter values represent the different weights for various components of generating the data. They are assigned some (arbitrarily) chosen values. γ denotes the vector representing these parameters.

(38) Data cost parameters

StemUsed	4
StemNotUsed	?
AffixUsed	1
AffixNotUsed	2
Extra	?

$$\gamma = \begin{bmatrix} 4 \\ 0 \\ 1 \\ 2 \\ 0 \end{bmatrix}$$

Conceptually, the set-up has five parameters as shown in (38), which means that the linguistic system incurs complexity in terms of five different components in order to generate the observed forms. We have to pay for (i) each stem letter used, (ii) each stem letter not used, (iii) each affix letter used, (iv) each affix letter not used, and (v) each extra letter from neither the stem nor the affix. Nevertheless, given the simple way in which the stems and affixes are derived, only three of these data cost components currently play an actual role in the algorithm; their weights are shown in (38). In further work, if we allow changes of the stem-affix boundaries, then there may be situations where a stem letter from the grammar is not used to generate the observed word (analogous to deleting a vowel or consonant from the stem's underlying form), or where an extra letter is needed (i.e., epenthesis). Until then,

the parameters StemNotUsed and Extra are inactive. We still choose to include them here for completeness.

For every target word, i.e., each of {*jump, jumps, jumping, jumped, jumper*} in (34), the goal is to use the stem and the relevant affix to generate it, and the algorithm keeps track of the cost associated. Formally, we define $u_i \subseteq t_i$ as the set of stem letters used from the stem t_i. It follows that $\bar{u}_i = t_i \setminus u_i$ is the set of stem letters not used from the stem t_i. Similarly, $v_j \subseteq x_j$ is the set of affix letters used from the affix x_j, $\bar{v}_j = x_j \setminus v_j$ is the set of affix letters not used from the affix x_j. e is the set of letters used from neither the stem nor the affix.

(39) Data cost for a target word $w_{ij} = \gamma^\top \begin{bmatrix} |u_i| \\ |\bar{u}_i| \\ |x_j| \\ |\bar{x}_j| \\ |e| \end{bmatrix}$

Take the target word *jumps* as an example. Its stem is *jump*, its affix is *s*, and so the data cost is as follows:

(40)

Data cost to generate *jumps* from stem *jump* and affix *s*

$$= \gamma^{\top} \begin{bmatrix} 4 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} 4 & 0 & 1 & 2 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$= 17 \text{ (sum of all entries in } \begin{bmatrix} 16 & 0 & 1 & 0 & 0 \end{bmatrix}^{\top}, \text{the second column in (41))}$$

Performing the same procedure for all target word forms in a given stemplex results in a data cost matrix, here the one for the JUMP stemplex:

(41) Data cost of the JUMP stemplex in (34)

	jump	jumps	jumping	jumped	jumper	
jump	Ø	s	ing	ed	er	
	16	16	16	16	16	(StemUsed)
	0	0	0	0	0	(StemNotUsed)
	0	1	3	2	2	(AffixUsed)
	0	0	0	0	0	(AffixNotUsed)
	0	0	0	0	0	(Extra)

Summing all entries in the data cost matrix gives 88 as the total data cost for the JUMP stemplex. The overall complexity of this stemplex, grammar plus data, is 85 + 88 = 173.

The algorithm computes the grammar cost and data cost in the way just described for all stemplexes. The total complexity of the stemplexes is the sum of all grammar costs and data costs.

3.4.3 Greedy optimization and minimum description length

The previous section presented in detail how complexity is computed. This section explains how the computed complexity is used for alignment and clustering.

With the stemplexes and total complexity initialized based on the input data, the algorithm undergoes an iterative, greedy process of minimizing the total complexity. This is achieved by iteratively merging two stemplexes. "Merge" here has nothing to do with Merge in the current Minimalist syntactic theory, but instead means: choose two stemplexes, compute the optimal alignment between them, and create a new stemplex based on this alignment, such that the total complexity decreases the most at the given particular iteration.

Union affixes and optimal alignment

The goal of alignment is to learn, for instance, that *jumps* is best aligned with *loves*, *jumped* with *loved*, and so forth. The key is the creation and comparison of *union affixes*. As an example, we consider the merging of the JUMP and LOVE stemplexes. The LOVE stemplex looks like the following, with a total complexity of 90 + 70 = 160; note that this verb is different from JUMP and belongs to the group with *e*-final stems:

(42) The LOVE stemplex

		love	loves	loving	loved	lover	⇐ TARGET WORD FORMS
STEM ⇒	lov	e	es	ing	ed	er	⇐ AFFIXES

(43) Grammar cost of the LOVE stemplex

$$\begin{aligned} & 5 \cdot (|lov| + |e| + |es| + |ing| + |ed| + |er| + 5) \\ =& 5 \cdot (3 + 1 + 2 + 3 + 2 + 2 + 5) \\ =& 90 \end{aligned}$$

(44) Data cost of the LOVE stemplex

	love	loves	loving	loved	lover	
lov	e	es	ing	ed	er	
	12	12	12	12	12	(StemUsed)
	0	0	0	0	0	(StemNotUsed)
	1	2	3	2	2	(AffixUsed)
	0	0	0	0	0	(AffixNotUsed)
	0	0	0	0	0	(Extra)

Total data cost = 70

The discussion is first in qualitative and intuitive terms, introducing the idea of union affixes. According to the algorithm (and our knowledge about English), the best alignment for the JUMP and LOVE stemplexes is in (45a); an affix is boxed if it differs from the union affix:

(45) Aligning JUMP and LOVE

a. Optimal alignment

Union affixes	**-e**	**-ed**	**-ing**	**-es**	**-er**
	jump-$\boxed{\emptyset}$	jump-*ed*	jump-*ing*	jump-$\boxed{s}$	jump-*er*
	lov-*e*	lov-*ed*	lov-*ing*	lov-*es*	lov-*er*

b. Suboptimal alignment

Union affixes	**-es**	**-ed**	**-ing**	**-es**	**-er**
	jump-$\boxed{s}$	jump-*ed*	jump-*ing*	jump-$\boxed{\emptyset}$	jump-*er*
	lov-$\boxed{e}$	lov-*ed*	lov-*ing*	lov-*es*	lov-*er*

To merge the stemplexes JUMP and LOVE into a new stemplex, the algorithm considers all 5! = 120 alignment possibilities. Given a particular alignment, the union affixes are

computed by taking the union of the affixes from the same column. For example, in the fourth column in (45a), the union affix *-es* is the union of the individual affixes *-e* and *-es*.[10] For this new JUMP-LOVE stemplex, the union affixes thus derived under this particular alignment are the new affixes, and are what counts towards the grammar cost of this new stemplex.

Because the grammar has been updated, the data cost for generating the observed forms for both JUMP and LOVE has to be recalculated based on the new affixes. For instance, in (45a), to generate *jumps*, the stem is *jump-*, and the union affix is *-es*. All four stem letters from *jump-* are used, but only one letter, *-s*, from the union affix *-es* is used. The unused union affix letter *-e* incurs a cost in the algorithm (the affix *-s* is boxed for differing from the corresponding union affix). Any other alignment deviating from (45a), such as (45b) with more boxed affixes, incurs a higher data cost than that of (45a). The alignment in (45a), with the lowest overall cost, is the best alignment for merging JUMP and LOVE.

Here are the quantitative details of the alignment just explained. Before merging, the total complexity of the JUMP is 173, and that of the LOVE is 160. The combined complexity of the two stemplexes is 173 + 160 = 333. Finding the optimal alignment is to merge the stemplexes such that the resultant new stemplex has the lowest complexity. Below are the cost details of the two alignments illustrated in (45).

(46) Data costs for aligning JUMP and LOVE

a. Optimal alignment

10. As always, we are dealing with bags of letters, not sets in the mathematical sense. So in a case with the same double letters in two individual affixes (hypothetically, *abb* and *bbc*), the union affix preserves the double letters (*abbc* as the union affix for this example).

	-e	**-ed**	**-ing**	**-es**	**-er**
jump-	jump-*Ø*	jump-*ed*	jump-*ing*	jump-*s*	jump-*er*
	16	16	16	16	16
	0	0	0	0	0
	0	1	3	2	2
	2	2	0	0	0
	0	0	0	0	0
lov-	lov-*e*	lov-*ed*	lov-*ing*	lov-*es*	lov-*er*
	12	12	12	12	12
	0	0	0	0	0
	1	2	3	2	2
	0	0	0	0	0
	0	0	0	0	0

Data cost = 92 + 70 = 162

b. Suboptimal alignment

	-es	**-ed**	**-ing**	**-es**	**-er**
jump-	jump-*s*	jump-*ed*	jump-*ing*	jump-*Ø*	jump-*er*
	16	16	16	16	16
	0	0	0	0	0
	0	1	3	2	2
	4	2	0	0	0
	0	0	0	0	0
lov-	lov-*e*	lov-*ed*	lov-*ing*	lov-*es*	lov-*er*
	12	12	12	12	12
	0	0	0	0	0
	2	1	3	2	2
	0	2	0	0	0
	0	0	0	0	0

Data cost = 94 + 72 = 166

(47) Grammar costs for aligning JUMP and LOVE

a. Optimal alignment

$$\begin{aligned} & 5(|jump| + |lov| + |e| + |ed| + |ing| + |es| + |er| + 5) \\ = \; & 5(4+ 3 + 1 + 2 + 3 + 2 + 2 + 5) \\ = \; & 110 \end{aligned}$$

b. Suboptimal alignment

$$\begin{aligned} & 5(|jump| + |lov| + |es| + |ed| + |ing| + |es| + |er| + 5) \\ = \; & 5(4+ 3 + 2 + 2 + 3 + 2 + 2 + 5) \\ = \; & 115 \end{aligned}$$

Examining the cost details in (46) and (47) reveals how the best alignment beats all other competing alignments. In terms of grammar cost, it is the differences in union affixes

in different alignments. As for data cost, the locus is the mismatches between union affixes and individual affixes.

The optimal alignment has a total complexity of 162 + 110 = 272. For the suboptimal alignment discussed, the complexity is 166 + 115 = 281. Both are lower than the pre-merging complexity of 333, but the complexity of the optimal alignment is the lowest among all 5! alignment permutations by saving 333 − 272 = 61. It is only 333 − 281 = 52 that this particular suboptimal alignment saves. The following table shows the cost saved of the best ten alignments.

(48) Costs saved of the best ten alignments for JUMP and LOVE

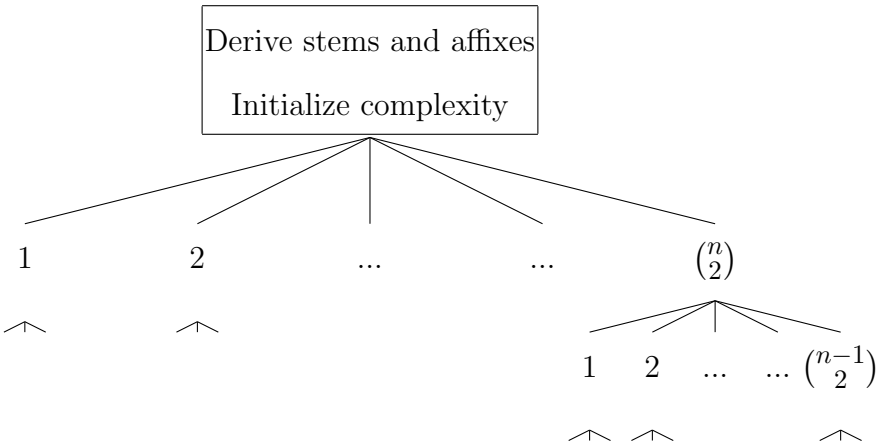
Rank	Cost saved
1	61
2	52
3	52
4	52
5	43
6	43
7	43
8	43
9	43
10	43

MDL-based iterative merging

With n stemplexes initially, each with k target word forms, there are $\binom{n}{2}$ distinct ways of picking a pair of stemplexes for merging. For each of these $\binom{n}{2}$ merging possibilities, there are $k!$ alignment permutations. The algorithm checks all $\binom{n}{2}k!$ options for the particular alignment in a specific merging option which lowers the total complexity the most.

Merging is performed iteratively as greedy MDL-based optimization. At each iteration, two stemplexes are merged with the best alignment between them, such that the grand total complexity for the overall data set is minimized. This is schematized as follows:

(49)



After the first iteration, the system has $n-1$ stemplexes left, and therefore the second iteration has $\binom{n-1}{2}$ merging possibilities. The iterative merging process ends when there is only one stemplex left in the system.

A remark on the greediness of the algorithm. Once two stemplexes are merged with their best alignment, the new stemplex stays as is. No un-merging or re-alignment is allowed. The algorithm does the best it can to lower complexity at each iteration. In brief, the algorithm does not look back, nor look ahead. Even if there may be *globally* less costly merging options down the road in future iterations that would require a *locally* suboptimal merging choice, the algorithm does not consider them. In fact, there is a practical, computational reason for the strict greediness: when merging involves a complex stemplex, if the algorithm took away the established alignments within that complex stemplex, then the number of (re-)alignment possibilities would grow exponentially over the factorials and the computation would take way too long.

The clustering effect

Finally, what is left to be accounted for is the clusters that mimic inflection classes for conjugation and declension. The key is the MDL-based and greedy nature of the algorithm. The more similar the paradigms, the earlier they merge in the iteration. From the perspective of machine learning, since all clustering algorithms employ some notion of distance among the objects in question, we can say that this paper proposes a measure for morphological similarity among paradigms in order to perform bottom-up, agglomerative clustering.

Let us for the moment consider the merging of the JUMP and WALK stemplexes.

(50) The grammar (stems and affixes) for merging JUMP and WALK

a. Before merging, with individual affixes for each stemplex:

jump-	-Ø	-ed	-ing	-s	-er
walk-	-Ø	-ed	-ing	-s	-er

b. After merging, with union affixes for the new stemplex:

jump-					
	-Ø	-ed	-ing	-s	-er
walk-					

The union affixes thus created for JUMP and WALK are identical to the individual affixes of both paradigms. This is doubly good in terms of complexity minimization. For the new grammar, one of the two (identical) sets of the individual affixes are effectively wiped out from the system. For the data cost, there is no increase. If the two sets of individual affixes were not identical, as is the case between JUMP (regular) and LOVE (with silent *e*) discussed in detail above, there would not be such advantages. The more similar paradigms attract one another.

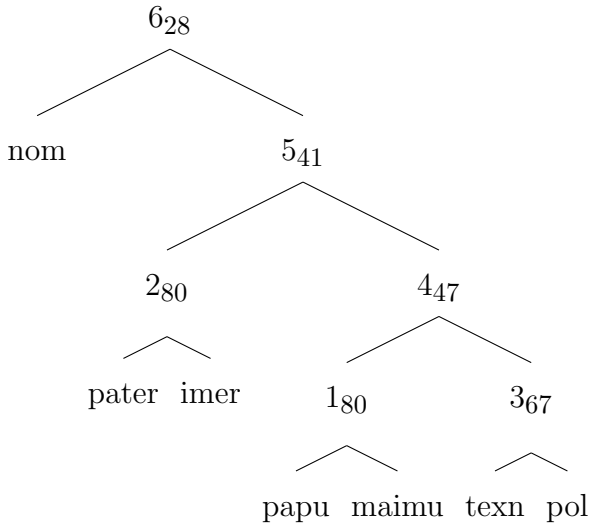
3.4.4 Back to Greek nominals

If we now have an algorithm which learns hierarchical patterns of inflection classes, then it should be interesting to try it with the Greek nominal data that Haspelmath (2002) considers. With everything unchanged in the algorithm, including all the grammar and data cost parameters, the seven Greek paradigms in (26) are analyzed as follows for alignment and clustering.

(51) Alignment results of the seven Greek nominal paradigms from (26)

mno	nomos	nomo	nomu	nomi	nomus	nomon
aeprt	pateras	patera	patera	pateres	pateres	pateron
eimr	imeras	imera	imera	imeres	imeres	imeron
appu	papus	papu	papu	papuðes	papuðes	papuðon
aimmu	maimus	maimu	maimu	maimuðes	maimuðes	maimuðon
entx	texnis	texni	texni	texnes	texnes	texnon
lop	poleos	poli	poli	poles	poles	poleon

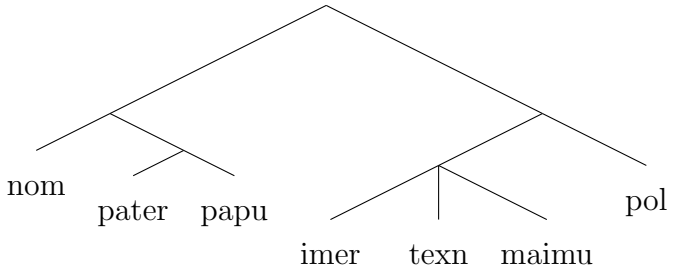
(52) Clustering results of the seven Greek nominal paradigms from (26)



The alignment in (51) does not match the original Greek data with morphosyntactic fea-

tures indicating the correct alignment in (26). As for clustering, the resultant tree looks quite distinct from what Haspelmath has come up with. For ease of comparison, Haspelmath's tree is simplified as follows:

(53) Haspelmath's inheritance hierarchy for the Greek nominals concerned



While the Greek results may be less satisfactory than the English ones, this is not entirely bad news for us. Understanding why we have obtained our results is instructive for further research. For alignment, one factor contributing to the undesirable results is that the algorithm is not (and should not be) able to see a strong generalization within the given data separating the masculine classes from the feminine ones. The singular nominative in the given *masculine* classes all ends with *-s*, but without it in the singular genitive. This pattern is the exact opposite in the *feminine* classes.

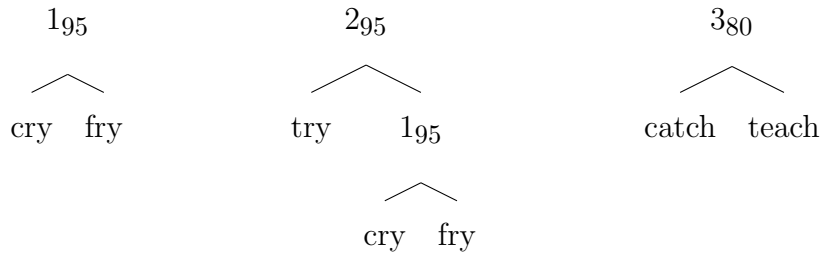
The crucial difference between Haspelmath's work and ours here is that Haspelmath knows *a priori* the alignment by the knowledge of morphosyntactic knowledge. His goal is to deal only with clustering. Indeed, as illustrated in (53), Haspelmath's inheritance hierarchical analysis has all masculine classes cluster together, the three leaves on the left. The other four leaves on the right are the feminine classes.

Juxtaposing Haspelmath's work and ours confronts us with the question of whether we assume prior assumption that, for instance, *jumping* and *loving* belong to the same morphological category in English. It is "yes", as in Haspelmath's discussion, if we assume knowledge of morphosyntactic features, their distribution, and all that (which non-computational linguists often do), but it is the "no" side that we would like to explore in this chapter and related work: we would like to explore how much we can learn if we remove assumptions which we are so used to and which we take for granted.

3.5 Results

The clustering results are visualized arboreally. With the 19 English verbal paradigms like those in (22) as input, the following are the first three merges.

(54) The first three merges



The first merge is between the CRY and FRY stemplexes, represented by the first tree in (54). The mother node says "1", which means this is the first merge. There is also the subscript "95", which tells us that this merge saves 95 units in complexity. The second merge is between a complex stemplex, created from merge 1, and the TRY stemplex. This merge also saves 95 units, which indicates that the order of merging among CRY, FRY, and TRY does not matter; this makes good sense, as they are morphologically identical. At the third iteration, the algorithm decides that it is best to merge CATCH and TEACH, which makes the total complexity drop by 80 units.

With 19 paradigms at the outset, there are 18 merges altogether ($n - 1$ merges for n input paradigms). The following table shows the cost saved at each merge.

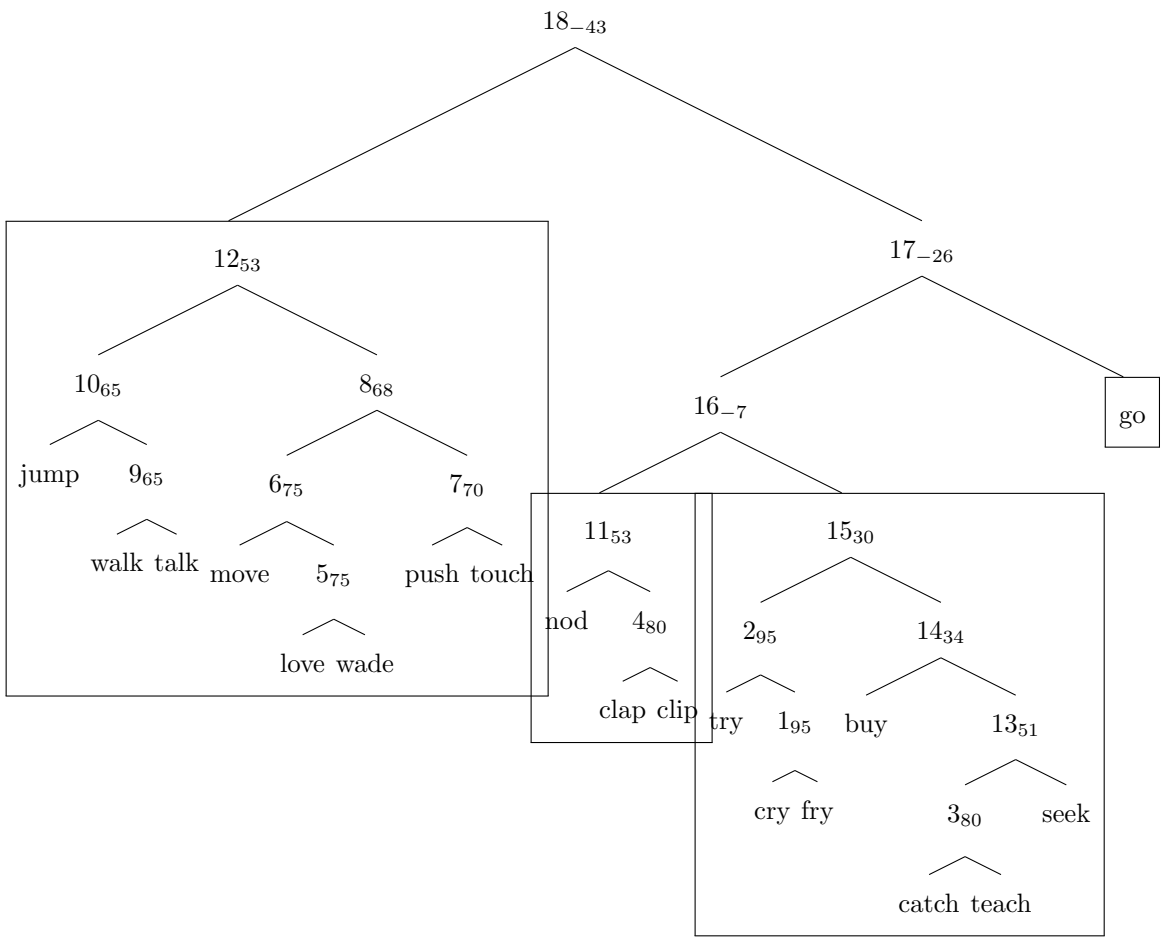
(55) Costs saved by merging

Merge	Cost saved	Merge	Cost saved
1	95	10	65
2	95	11	53
3	80	12	53
4	80	13	51
5	75	14	34
6	75	15	30
7	70	16	-7
8	68	17	-26
9	65	18	-43

Importantly, the costs saved by successive merges are decreasing, or at least non-increasing for ties between two merges. This is the case because the algorithm aims at decreasing the total complexity as quickly as possible. From the 16th merge onwards, the costs saved are negative. The algorithm can no longer actually decrease the total complexity. Nonetheless, algorithm does not stop until there is only one stemplex left, and the best it can do is to increase complexity the least. In other words, from merge 16 through 18, the total complexity increases as little as possible.

For more interpretation of the clustering results, let us examine the complete tree:

(56) Clustering results



From (56), one can visually identify English 'conjugation groups' such as the following; they are reminiscent of Bloch's (1947) grouping of English verbal inflection classes:

(57) Some English 'conjugation classes'

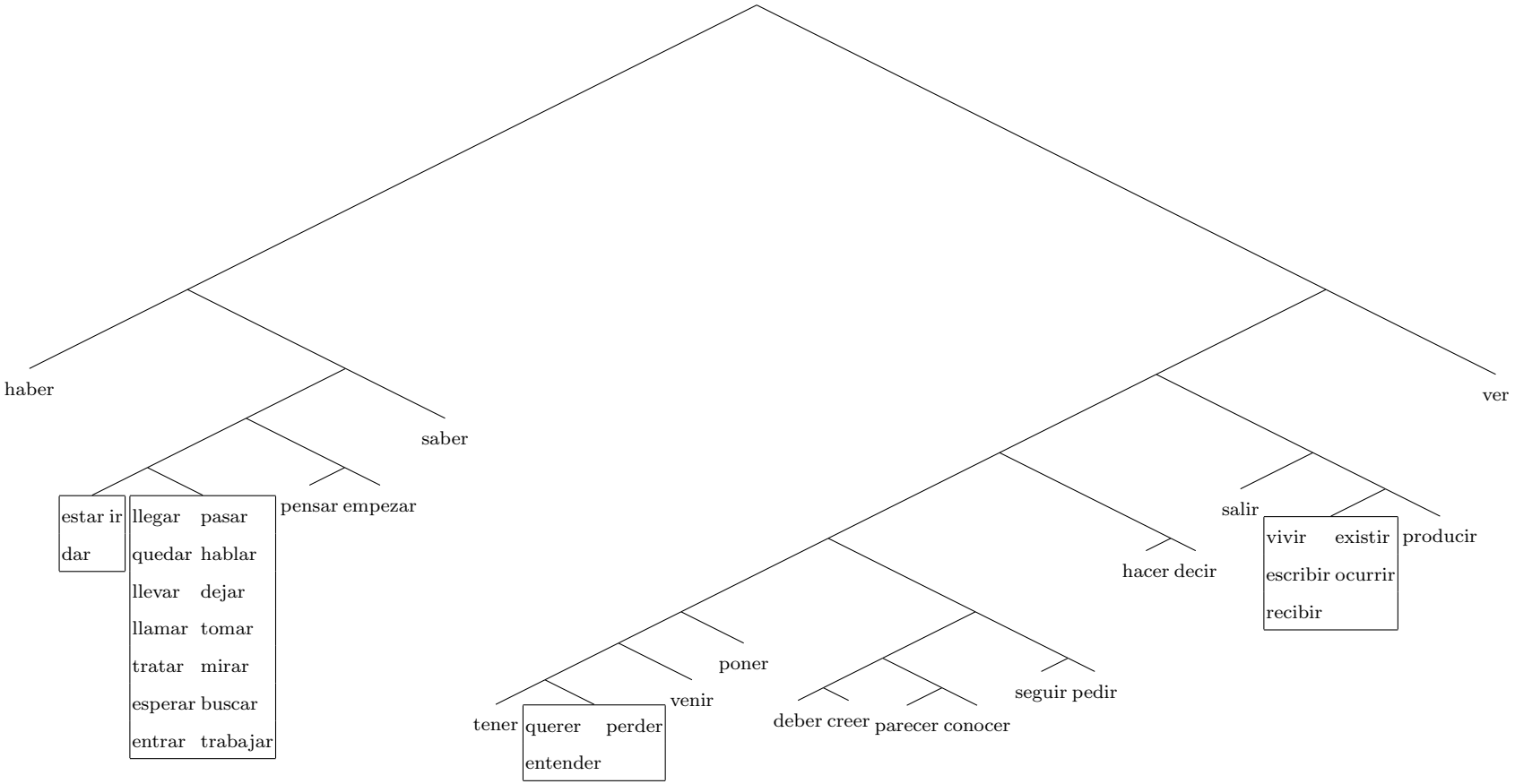
Property	**Paradigms**
Regular	JUMP, WALK, ...
Silent 'e'	LOVE, MOVE, ...
Consonant doubling	NOD, CLAP, ...
'y/ie'	TRY, CRY, ...
'es' for 3sg	PUSH, TOUCH, ...
'{a,o}ught' for past	BUY, SEEK, ...
Suppletive	GO, ...

At another level, the negative saved costs in (55) can be interpreted as indicators of more general, major clusters (Haspelmath's macro-classes). This is *a* way of interpreting them, and in this particular one, they tell us where the algorithm should not have merged, so to speak. In (56), the boxes indicate the main clusters under this interpretation. The groupings appear to reflect suppletiveness or the amount of 'morphophonological alternations' involved in the paradigms.

Given the generality of the algorithm and to point to directions of further work, we briefly examine in the following the preliminary clustering results of Spanish verbal paradigms alluded to from time to time in the discussion above.

We take as input data the 50 most common verbs in Spanish, conjugated in the present indicative; each verb lexeme has six inflected forms, such as *hablo-hablas-habla-hablamos-hablaís-hablan* for *hablar* 'to speak'. The algorithm takes the given alignment in the input data as is (section 3.4.4) and assumed the same cost parameter γ (38). The resultant clustering tree, with merge numbers and costs removed for simplicity of exposition, is as follows:[11]

11. The dataset and output files are also found here (showing a partial tree here for reasons of space): `https://github.com/JacksonLLee/morph-align-cluster`


haber
saber
ver
estar ir
dar
llegar pasar
quedar hablar
llevar dejar
llamar tomar
tratar mirar
esperar buscar
entrar trabajar
pensar empezar
salir
vivir existir
escribir ocurrir
recibir
producir
hacer decir
poner
venir
tener
querer perder
entender
deber creer
parecer conocer
seguir pedir

The boxes at some of the leaves in the tree above indicate that the paradigms in the same box are determined to be morphological identical, i.e., having the same inflectional pattern; the largest box is the one with non-stem-changing -AR verbs such as *llegar, quedar*.

3.6 Conclusion

We have proposed a language-independent clustering algorithm which learns structure across morphological paradigms for inflection classes and their hierarchical patterns. The work is a step forward, built on top of Haspelmath's illustration of Greek nominals; we have also shown results from other languages such as English and Spanish. The starting point of this chapter is the availability of morphological paradigms like the English verbs in (22). A question lurking in the background is the following: Where do these paradigms come from in the first place? This is the subject of the next chapter.

CHAPTER 4

PARADIGM INDUCTION AND ALIGNMENT

4.1 Introduction

The previous chapters on stem extraction and paradigm similarity assume that morphological paradigms are given to the analyst. In particular, for paradigm similarity, the morphological paradigm datasets are neatly in the form of tables, where each row is the paradigm of one lexeme, and each column is the word form of a particular morphosyntactic feature. This chapter asks the following question: how we do obtain tables of morphological paradigms, just like those we have used in the previous chapters? To answer this question, this chapter takes the approach of unsupervised learning of morphological paradigms from raw text; the text or language in question has orthographic word boundaries (e.g., spaces), as we are not handling word segmentation in this dissertation. Specifically, this chapter discusses two tasks: paradigm induction and paradigm alignment.

When we discussed structure within a paradigm and across paradigms in the previous chapters, we assumed that a full paradigm table, as in (58) below, is available to a computer program.

(58) A full paradigm table (e.g., English verbs)

talk	talks	talked	talked	talking
take	takes	took	taken	taking
move	moves	moved	moved	moving
⋮	⋮	⋮	⋮	⋮

A full paradigm table has three defining properties which the exploration of paradigmatic structure in previous chapters makes use of:

(59) Properties of a full paradigm table (as in (58)):

a. Each row contains word forms from one lexeme, e.g., TALK (first row above) as opposed to TAKE (second row).

b. Each column contains word forms from the same morphosyntactic category, e.g., *talks, takes, moves* for third person singular (second column) as opposed to *talked, took, moved* for simple past (third column).

c. The table is complete with no empty cells.

Paradigm induction is about how to learn the paradigms, such as the individual rows in (58), from raw text. For instance, in English, the rows learned can be {jump, jumps, jumped, jumping} as one row, or {create, creates, created, creating} as another. While each row is a set of word forms that belong to a lexeme, paradigm induction is not responsible for the way in which one row of word forms should be aligned with another row – this is where paradigm alignment plays the role of figuring out what the columns should be. Ideally, the results of paradigm alignment are in line with the morphosyntactic features of the word forms involved. Using the English examples here, jump and create should be placed in the same column in an English paradigm table, whereas jumps and creates in another. It is when both paradigm induction and paradigm alignment are solved that we obtain a paradigm table such as (58).

4.2 Learning morphology from raw text

The use of raw text in learning paradigm tables begs the following questions: How do we know (i) which and how many rows of paradigms exist in the raw text data, (ii) which and how many columns of morphosyntactic categories are there?

These questions bear closely on language acquisition research, for the question of how the learner of a language acquires its morphological structure from the natural linguistic input. In the present context, learning morphology from raw text by a computer program is different from the human naturalistic setting. Raw text has linguistic data transformed into a machine-readable form with word-level segmentation given, very much like the text of this

dissertation with which the reader is presented. Although word segmentation is given, the task of morphological paradigm learning is still faced with empirical challenges associated with the "poverty of the stimulus" (Chomsky 1980; Pullum and Scholz 2002; Niyogi 2006). First, there is no negative evidence, in the sense that humans acquiring their first language do not normally obtain explicit feedback for producing ungrammatical utterances. Second, there is not sufficient positive evidence in the linguistic input. This point is widely discussed in terms of data sparsity in the computational literature (e.g., not all n-grams of possible word sequences are observed in any finite size of corpus; Jurafsky and Martin 2006), and in terms of (morphological) productivity in the more traditional linguistic literature (Hockett, 1960; Bauer, 2001). For morphological paradigms, for instance, it is entirely possible to observe only {jump, jumped, jumping} but jumps in a particular text. Another observed, incomplete paradigm may well be {loves, loving}. Even if a computer program succeeded at the paradigm induction problem by learning these paradigms, there would still be the issue of paradigm alignment, e.g., aligning "loving" with "jumping", while not aligning "loves" with either "jump" or "jumped".

4.3 Paradigm induction

The paradigm induction problem is to look for morphologically related words by lexeme in a text corpus. These related word forms are the basis for the rows in a paradigm table. There has been a good amount of work on inducing paradigms from a corpus text in one way or another (Yarowsky and Wicentowski, 2000; Goldsmith, 2000, 2001, 2006; Schone and Jurafsky, 2000, 2001; Baroni et al., 2002; Creutz, 2003; Creutz and Lagus, 2005; Zeman, 2008; Dreyer, 2011; Dreyer and Eisner, 2011; Borg and Gatt, 2014). This dissertation builds on the results from *Linguistica* by Goldsmith (2000, 2001, 2006).

Linguistica by Goldsmith (2000, 2001, 2006) takes a sizeable wordlist of an unknown language as the input and induces groups of paradigms with the associated stems and affix

patterns by the minimum length description principle (Rissanen, 1989). These groups are known as *signatures*. For example, {Ø, s} is a morphological signature very likely to be induced in any sizable English datasets, with possible associated stems such as *walk-, jump-* (which entails that the word forms *walk, walks, jump, jumps* occur in the data).

4.3.1 Results from Linguistica 5

Using the Brown corpus (about 50,000 word types from one million word tokens) for written American English, *Linguistica 5* finds over 300 morphological signatures. Those with the most associated stems are shown in the screenshot in Figure 4.1; the signature {Ø, ed, ing, s} is highlighted, with its associated stems displayed on the right.

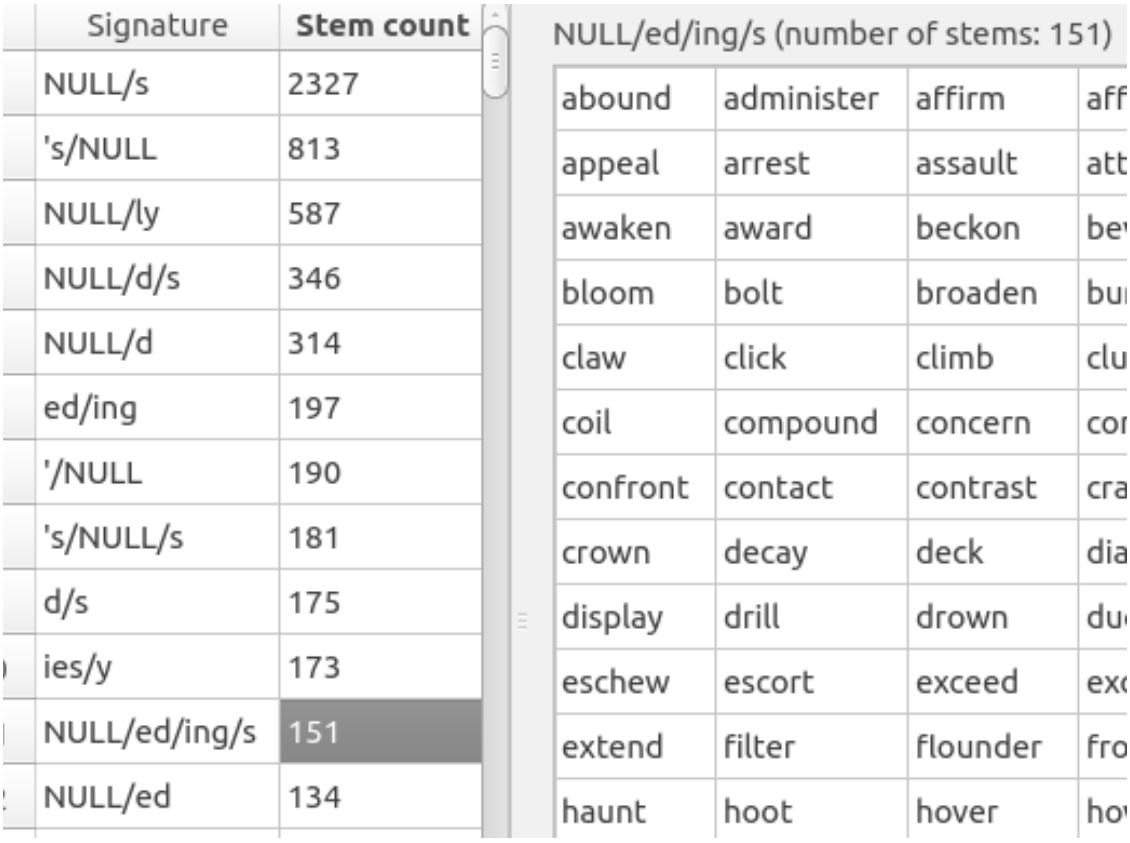


Figure 4.1: Signatures with the most stems in the Brown corpus

As an example, with Brown corpus (Kučera and Francis, 1967) as the input corpus text

from which an English wordlist is derived, *Linguistica* induces the following top ten signatures ranked by the number of stems associated with each signature:

(60) English signatures induced from Brown corpus by *Linguistica*

Signature	Sample paradigm	Stem count
NULL-s	airport airports	2341
's-NULL	barber's barber	784
NULL-ly	abnormal abnormally	658
NULL-d-s	assume assumed assumes	362
NULL-d	choke choked	317
'-NULL	cousins' cousins	187
ies-y	faculties faculty	186
's-NULL-s	actor's actor actors	184
NULL-ed-ing-s	allow allowed allowing allows	184
ed-ing	darkened darkening	184

In (60), a row consists of a SIGNATURE, i.e., a unique set of affixes. The most common signature by stem count is NULL-s. A sample paradigm is *airport*∼*airports*, where the stem is *airport-*. The *Linguistica* program finds 2,341 stems such as *airport-* which are associated with the signature NULL-s. In other words, there are 2,341 paradigms with this exact same affixal pattern with NULL-s, and another 784 paradigms with the signature 's-NULL, and so forth. Although some signatures share identical or similar affixes, *Linguistica* treats all signatures as completely distinct from one another and makes no attempt to find relationships among them.

An examination of these corpus-derived paradigms reveals the contrast between them and the paradigms in a full paradigm table, and demonstrates how challenging it is to properly construct a paradigm table from a text corpus alone. Consider English verbal paradigms. For regular verbs, there are four distinct inflected forms, e.g., *jump*∼*jumped*∼*jumping*∼*jumps*

for JUMP. The signature NULL-ed-ing-s corresponds to this verbal inflectional pattern in English, and happens to be in (60) as one of the most common signatures. But there are two issues here, as (60) shows. First, despite the large sizes of corpora, a lexeme is typically observed to be inflected only in some but not all of its possible forms. The issues are those of data sparsity and strongly skewed distributions among lexemes (Baayen, 2001). The signature ed-ing in (60) is a case in point, with *darkened*∼*darkening* being one of the associated paradigms: if both *darken* and *darkens* were in Brown corpus, then this DARKEN paradigm would be in the signature NULL-ed-ing-s. The second issue has to do with morphophonology, for both phonological and orthographic alternations. Both NULL-d-s and NULL-d are among the most common signatures in (60), and they are clearly verbal paradigms. What is shared by the paradigms associated with both signatures is the silent 'e' in the infinitival form. We would want to put these paradigms from NULL-d-s and NULL-d (with silent 'e') with those from NULL-ed-ing-s and others (without silent 'e') into a single paradigm table, because they are all verbal paradigms.

Now that we have a way to learn morphological paradigms from a text corpus as signatures, we have the rows of a paradigm table, where each row is the word forms of a lexeme. What is missing is the alignment by columns. For instance, given the signatures NULL-ed-ing-s and NULL-d-s, both of which are presumably for English verbal paradigms, we would want the NULL word forms from both signatures to belong to the same column in the paradigm table, "ed" should go with "d", "s" from both signatures should be in another column, and finally the "ing" word forms from the first signature should not be aligned with any paradigms in the NULL-d-s signature. The next section discusses this paradigm alignment problem.

4.4 Paradigm alignment

In this dissertation, the approach to solving the paradigm alignment problem is inspired by how we appear to have acquired morphosyntactic knowledge naturally.

Intuitively, a simple subset principle might seem to be able to do this: if signature A has all affixes in signature B, and if signature A has more affixes than signature B, then collapse signatures A and B by keeping A, moving all paradigms from B to A, and removing B. Both over-collapsing and under-collapsing arise as potential issues out of this strategy. On over-collapsing, due to (accidental) syncretism, collapsing NULL-s (with both nominal and verbal paradigms in English) and the verb-only NULL-ed-ing-s signature, though obeying this subset principle, would be problematic. As for under-collapsing, because of surface affixal differences between NULL-d-s and NULL-ed-ing-s, these two verbal signatures cannot be straightforwardly collapsed by the subset principle alone.

As Goldsmith (2009) points out, aligning signatures and their resulting morphological paradigms is a challenge. The crux of the problem appears to be the lack of some kind of syntactic information. *Linguistica* takes as the input a plain wordlist with no annotations or labels whatsoever. If we had, say, part-of-speech annotations (which Chan (2006) assumes in paradigm induction, for instance), then both over-collapsing and under-collapsing would be mitigated. The paradigms in NULL-s would presumably be distinguished as verbal and nominal paradigms. All paradigms belonging to the same part of speech, irrespective of signatures, could be forced into the same paradigm table, with the correct morphological alignment computed by an algorithm similar to the one described in the previous chapter on paradigm similarity.

The goal of this dissertation is to induce morphological structure by a fully unsupervised approach; assuming annotations such as part-of-speech labels would be a clear violation. The following sections describe the induction of morphosyntactic alignment across words in a raw text, and how such learned morphosyntactic knowledge is used in solving paradigm

alignment.

4.4.1 Unsupervised word category induction

The problem of inducing morphosyntactic categories for words from a text corpus is very much the problem of unsupervised word category induction (Christodoulopoulos et al. (2010)).

What are morphosyntactic categories? As a first approximation, they are what is known as lexical categories, or parts of speech, or word classes, e.g., nouns, verbs, and so on. There are eight broad categories of this type widely recognized for English: nouns, verbs, pronouns, adjectives, adverbs, prepositions, conjunctions, and interjections. This means that all the word tokens in a large corpus of English can, in principle, be assigned one of the members from this much small set of categories. One can make finer distinctions among these broad categories. For instance, the Brown corpus (Kučera and Francis, 1967) uses a part-of-speech tagset with more than 80 distinct tags; for verbs, VBD labels verbs in simple past tense and VBG verbs in the *-ing* form. In this dissertation, we use "morphosyntactic categories" as a cover term to mean both broad and narrow types of categories.

Morphosyntactic categories have been treated in various ways in the literature. In the more traditional linguistic studies, the focus is to model the syntactic and semantic similarities and differences across the (broad) morphosyntactic categories. In generative studies, early attempts include the parametric ones using different features of [± X] and the natural classes thus formed (Chomsky, 1970; Jackendoff, 1977; Déchaine, 1993). However, Baker (2003), among others, observes that formal approaches such as these do not seem to have had strong ramifications; indeed, regardless of theoretical frameworks, current linguistic practice appears to be already content with the more traditional and descriptive labels such as N(oun) and V(erb) and therefore speaks of NPs and VPs.

In the computational literature, works of our interest are under the rubric of part-of-speech (POS) tagging. The focus is to assign a part-of-speech (equivalent to morphosyntactic

category in our sense) label. These studies can be divided into those using supervised learning techniques and those using unsupervised ones. As discussed above (sec 1.4.2), the main difference is what the input looks like, and how much is assumed to be given. In actual practice, most works in this area are supervised in one way or another. If a learning system assigns POS tags in a truly unsupervised manner, then there should only be a raw text as the input, and the system must have no language-specific knowledge at all. If this is the case, then there is no way to obtain outputs with such labels as nouns or verbs. For practical purposes, this is not desirable. A good amount of work on POS tagging is essentially what Goldwater and Griffiths (2007) call POS disambiguation, which means there is a dictionary wordlist with each word associated with multiple possible POS labels or the system a priori knows the list of POS labels each with some prototype words (Merialdo, 1994; Banko and Moore, 2004; Wang and Schuurmans, 2005; Smith and Eisner, 2005; Haghighi and Klein, 2006). Fully unsupervised approaches to POS tagging typically make use of the distribution of words in an unannotated corpus and define a distance metric between words for clustering, possibly with additional information such as morphology learned from the raw corpus (Schütze, 1995; Clark, 2000, 2003). Despite the attractiveness of fully unsupervised learning techniques such as clustering, it might not be straightforward to evaluate results as gold standards do not normally exist in the form of (unlabeled) clusters.

In this dissertation, we employ a fully unsupervised graph-theoretic approach to inducing morphosyntactic categories, drawing from Goldsmith and Wang (2012). In the discussion that follows, we focus on the intuition of how Goldsmith and Wang's method can take a text corpus and use distributional information across words to help us infer that, for example, *jumps, takes, walks* belong to a morphosyntactic category where *jump, take, walk* are excluded.

Given an unannotated corpus, the distributional information is derived from the word-based trigrams of the corpus. To the extent that morphosyntactic and semantic information

is reflected by the distribution of the words, using n-grams is a valid approach. Moreover, n-grams are readily available in an unannotated corpus, which makes unsupervised learning possible with minimal language-specific assumptions. This unsupervised learning approach of using distributional information only in some unlabeled linguistic data has also been employed in other areas of linguistics such as phonology, see, e.g., Riggle (2011); Goldsmith and Riggle (2012).

The following are the ten most frequent trigrams in the Brown corpus:

(61) The most frequent trigrams in the Brown corpus

Rank	Trigram	Count
1	, and the	662
2	one of the	403
3	the united states	328
4	, however ,	321
5	, in the	266
6	, he said	257
7	as well as	238
8	, it is	234
9	, and he	225
10	of course ,	220

Each word token is associated with three sets of trigrams. Take *beginning* in the sequence "In the beginning God created..." that begins the Bible. The first trigram has *beginning* as the third word, i.e., (In, the, *beginning*). The second trigram has *beginning* in the middle, (the, *beginning*, God). The third one has *beginning* at the end, (*beginning*, God, created). Information of this sort for all word types in the corpus is collected; intuitively, two word types such as modals *would* and *must* have similar patterns of neighboring words, which would suggest that *would* and *must* share similar neighboring words and therefore similar

morphosyntactic distributions.

Then, a series of graph-theoretic operations compute the distributionally (and morphosyntactically) most similar words for each word types. First, a graph of word similarity for all pairs of word types is computed based on the number of shared word n-gram contexts. We compute the most significant eigenvectors of the normalized Laplacian of the graph. Each word is embedded in $\mathbb{R}^k$ based on the coordinates derived from the k eigenvectors. A new graph of word similarity is obtained based on the Euclidean distance of the word coordinates. Words in this resultant graph are connected to one another in such a way that corresponds to syntactic neighborhood. For instance, the word "the" likely has other articles or determiners such as "a" and "an" as syntactic neighbors that occur in syntactically similar positions. Using the Brown corpus Kučera and Francis (1967), several syntactic neighbors for the word types "the", "would", and "after" are in (62).

(62) Syntactic neighbors

Word	Most similar words
all	such some after than like about even before then
over	about on into up at like out from with
its	our their my this his such the some these
before	after like about then that than when all what
had	has have did said do made first like who
to	in at for by from on of into with
only	also even made now no more any so one
has	had have did said first do who like)
then	now before what when that like but after than
them	me him her it you up out do my

Importantly, the syntactic neighbors of a given word are themselves word types in the given dataset. The interconnectedness of words in the syntactic neighborhood results calls

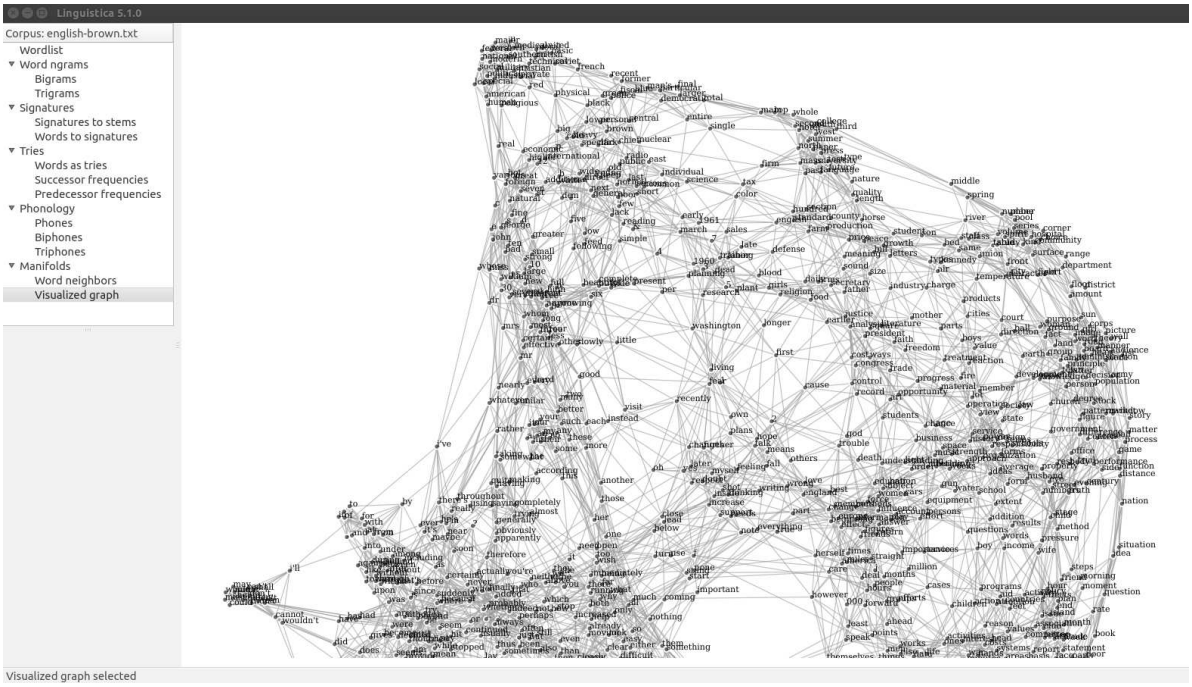


Figure 4.2: Syntactic word neighborhood network in *Linguistica 5*

for network visualization. This can be done in *Linguistica 5* as one of the key new features. Figure 4.2 shows a screenshot of *Linguistica 5* displaying the syntactic word neighborhood network for the most frequent 1,000 word types in the Brown corpus, as rendered by the force-directed graph layout in the JavaScript D3 library Bostock et al. (2011). Figure 4.3 zooms in for the cluster of words that would be categorized as modal verbs such as "could", "would", and "must".

With induced knowledge analogous to word categories in natural language, results of unsupervised morphological learning could be improved. For instance, morphophonology could be learned. Induced morphological signatures such as {Ø, ed} (*walk-walked*) and {Ø, d} (*love-loved*) could be aligned for orthographic allomorphy across signatures (words with *ed* and *d* belonging to the same word category in this case).

A word in the left-hand column in (62) above can be one of the most similar words for another word. For example, the first row has *all* and its nine most similar words, while the fourth row has *before* with *all* being one of the most similar words of *before*. This sort

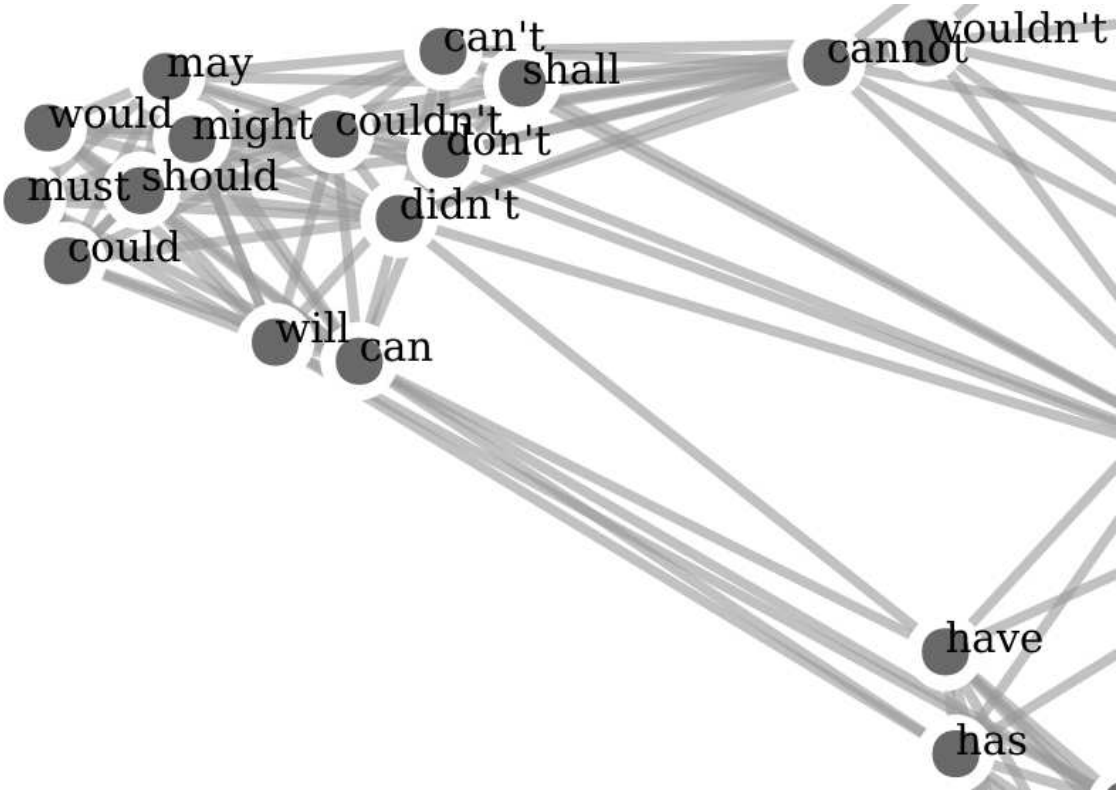


Figure 4.3: Zooming in Figure 4.2 for modal verbs

of intertwining relationship among words can be more meaningfully visualized as a graph, where a node represents a word and is connected by an edge to the word's most similar words. For example, take *would* as our seed word, and ask what the five distributionally most similar words are. For each of these most similar words, we ask again what its five most similar words are, and we repeat this process twice; in other words, we generate a graph with three generations of nodes seeded by *would*. The graph is as followed:

In the graph (4.4) above, the orange node *would* is the seed with five blue nodes connected to it. These five nodes represent *may, will, could, do, to*. Each of these five blue nodes is further connected to five nodes. If these latter nodes are not one of those already present (either orange or blue) in the graph, then they will be yellow. With five blue nodes each connected to another five nodes, there could be 25 ($= 5 \times 5$) yellow nodes in principle, but there are only 11 in the graph above. This means that some of the blue words (and the

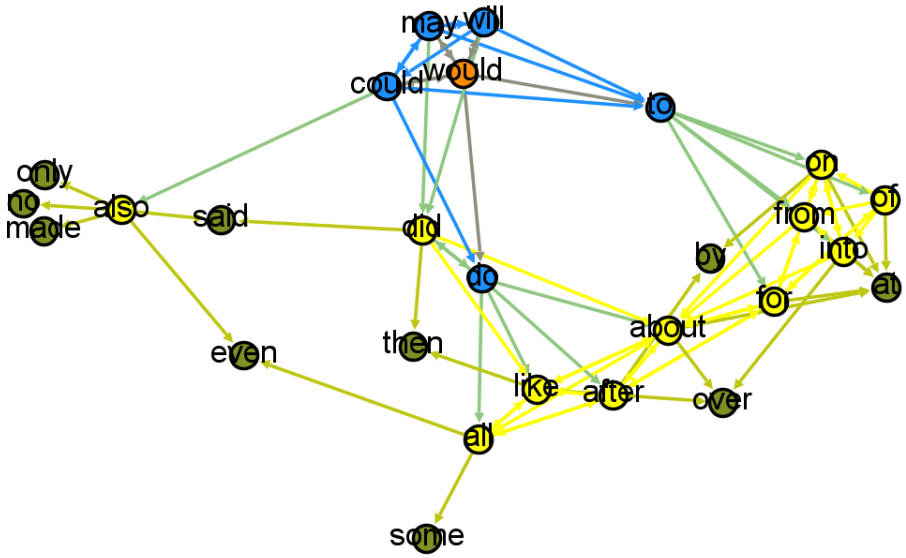


Figure 4.4: *would* as seed word, 3 generations of 5 most similar words

orange *would*) are highly connected among themselves. This translates into the observation that these words together are distributionally very similar; in this particular case, they are words typically followed by a bare verb form. If we move on to the next generation of nodes, the same observation applies. With 11 yellow nodes, we might expect 55 ($= 11 \times 5$) connected nodes (the green ones in the graph above), but there are only 10. This is because the yellow nodes are strongly inter-connected; most of them are prepositions typically followed by a noun.

The graph for *would* above shows how the Goldsmith-Wang algorithm clusters words by what is analogous to morphosyntactic categories. This graph of *would* can be seen as a

snippet of a much larger picture, both literally and computationally. The following shows the graph for the 1,000 most frequent word types in the Brown corpus:

Figure 4.5: Graph of the 1,000 most frequent words from the Brown corpus

In this particular two-dimensional rendering of the 1,000-node graph, there are portions which stick out prominently, much like the legs of an amoeba. If we zoom in onto the "northwest" corner of this graph and examine the nodes, we see what is desirable with respect to morphosyntactic categories:

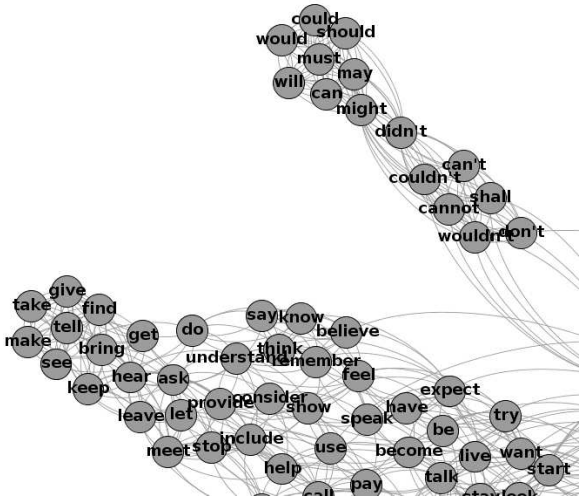


Figure 4.6: English modal verbs and infinitival verbs as separate clusters

Modals and auxiliaries cluster together, and so do verbs in their infinitival form; see (4.6). Clusters of nodes of this sort are what we look for. Computationally, they are the minor subgraphs ("minor" in the sense of having a much smaller number of nodes compared to the major subgraph) which have densely connected nodes within themselves but are connected to the major subgraph with a relatively small number of edges.

Finally, to map words in a graph to their induced word categories, we apply a community detection method, such as the Louvain algorithm (Blondel et al. 2008). Community detection is a common technique in network analysis, where the interest is how entities in a network may cluster together. Given the discussion about words' morphosyntactic behavior and how they may cluster together, it would seem that community detection is appropriate for our purposes of inducing word category induction. (4.7) visualizes the result of community detection by coloring a graph of the same top 1,000 most frequent words from the Brown corpus by their induced categories.

Now that we have an unsupervised method to induce word categories from a raw text, the next section discusses how to combine morphological knowledge from paradigm induction and morphosyntactic, distributional knowledge in order to solve paradigm alignment.

4.4.2 Combining morphological and distributional knowledge

To morphosyntactically align morphological paradigms, a simple algorithm based purely on the learned word categories is as follows:

- For each signature (e.g., NULL-s), the goal is to find the word category assignment for each affix ("NULL" and "s").
- The word category assignment for an affix is based on the associated word forms in that signature. For each of these word forms, we know the learned word category by the graph-based approach in the previous section.

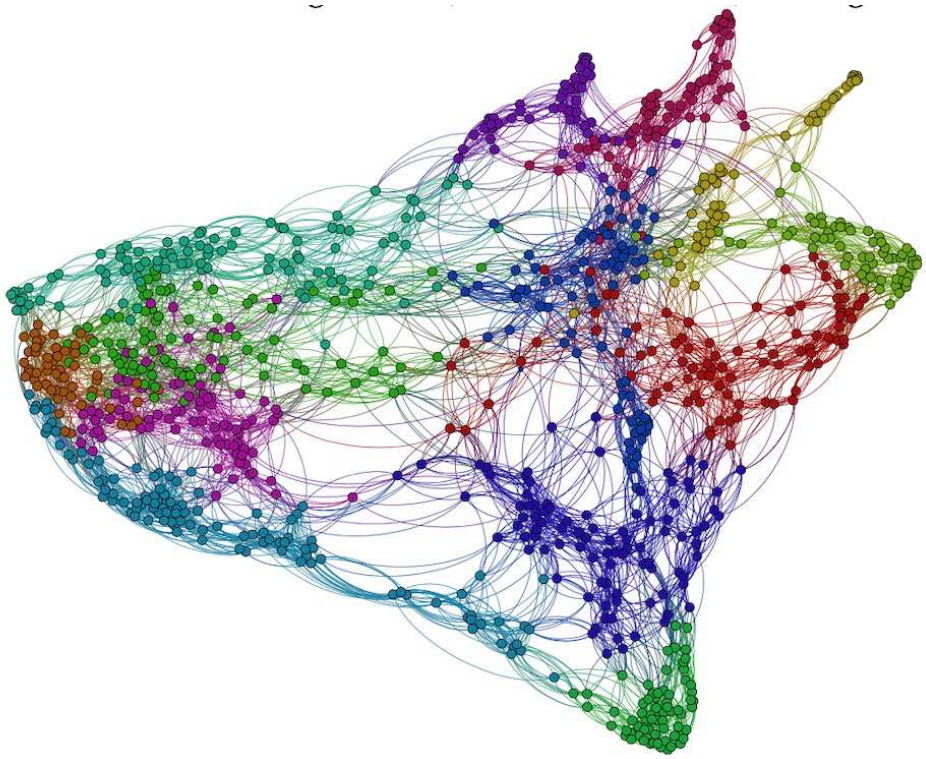


Figure 4.7: Graph of the 1,000 most frequent words from the Brown corpus, colored by induced word categories

- The word forms for a given affix would likely disagree for the overall word category assignment. A simple resolution would be by simple majority: The word category that has the most number of word forms for that affix is the final word category assignment for that affix as well.

With all affixes of all signatures assigned a word category, we can visualize the result in a table:

(63) Aligning morphological signatures

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
NULL-s			s	NULL											
's-NULL					's		NULL								
NULL-ly					ly			NULL							
NULL-d-s			s						d		NULL				
NULL-d									d						
ies-y							y			ies					
NULL-ed-ing-s			s						ed		NULL			ing	
's-NULL-s			NULL												
'-NULL											NULL				
ed-ing															
d-s					s				d						
e-ed-es-ing	ing								ed		e			es	
NULL-ed															
NULL-ed-ing		NULL													
d-r		r							d						
e-y		e			y										
d-r-rs									d	r					
NULL-ing									NULL						
ng-on												ng			
NULL-ed-s											NULL				
e-ed-ing															
'-g									g						
ng-on-ons			on												
ce-t										ce					
NULL-al-s							NULL	al							
NULL-es															
ed-ion							ion								
NULL-al			NULL												
NULL-ly-s		NULL													
NULL-y															NULL
's-NULL-s-s'		NULL	s												
NULL-n															
m-t															
d-rs									d						

In (63), the columns represent the word categories 1, 2, and so on. Each row is for a signature, and its affixes are indicated in the row for which word category it is assigned to. Some of the columns would appear to be desirable, e.g., word category 8 for what would

seem to be the past tense or past participle words with "-(e)d". Note that this method of alignment is based only on the distributional knowledge. Further work may incorporate knowledge of the surface string similarity to improve the result, e.g., the "(i)ng" affixes should presumably be aligned to the same column.

CHAPTER 5

TOWARD ACQUISITION MODELING

5.1 Introduction

Two themes have emerged from this dissertation research:

- Learning linguistic structure from raw data, as exemplified by the several topics in computational morphology studied in this dissertation
- Reproducible, accessible, and extensible research, as operationalized by modern software engineering practice

While there are multiple directions in which this research can continue, in this chapter I sketch one of them of particular interest to me, and describe some of the related, ongoing work.

Across the topics of computational morphology discussed in this dissertation—stem extraction, paradigm similarity, as well as paradigm induction and alignment—the data being used starts from the more linguistic structured to the less: first morphological paradigms given by lexeme, and then unannotated raw text. What is at odds is that learning linguistic structure from raw text (such as the Brown text used extensively in this work, or Wikipedia, or another large text dump) is *not* how humans acquire language. Humans acquire language incrementally, in two important senses: (i) there is a time dimension, such that the linguistic input data becomes available continuously over time, and (ii) the input data increases in complexity for both its content and grammatical structure. Feeding linguistic data as a single batch to a computer program, as has been done in this dissertation so far, eliminates these two incremental aspects of language acquisition. My ongoing work attempts to fill this gap, by modeling the learning of linguistic structure using naturalistic language acquisition

data, and by building tools related to this line of work for reproducibility, accessibility, and extensibility.

5.2 Working with CHILDES data programmatically

Natural language data often come in the form of conversations transcribed in some specific format for the purposes of linguistic research and other domains that require a consistent representation of conversational data. A very commonly used format is the CHAT transcription format (MacWhinney, 2000). CHAT (Codes for the Human Analysis of Transcripts) is the transcription format particularly developed for CHILDES (Child Language Data Exchange System) for language acquisition research. As CHAT is well documented and can have very rich annotations, it is also used more generally outside the field of language acquisition, for areas such as conversational analysis, corpus linguistics, and clinical linguistics.

Research using data in the CHAT format necessitates tools for extracting information and doing analysis in an efficient and automatic manner. This is particularly relevant for the computational modeling of language acquisition, a growing field of study across linguistics, psychology, and computer science (cf. Alishahi 2010; Villavicencio et al. 2013). The CHILDES project has the associated tool CLAN (Computerized Language Analysis), a widely used toolkit with a graphical user interface which facilitates both transcription and analysis of conversational linguistic data. As a standalone computer program, however, CLAN does not straightforwardly allow customized manipulation and analysis of CHAT transcripts that deviates from the functionalities directly provided by CLAN itself. To this end, a solution would be to come up with something that parses CHAT transcripts and allows researchers to devise any tools and programs for their purposes. For instance, it would be desirable to be able to parse CHAT data, perform computational and statistical analyses, as well as visualize data and results all in one single system.

Indeed, the Python-based NLTK (Natural Language Toolkit; Bird et al. 2009) has a

CHILDES corpus reader (by Tomonori Nagano and Alexis Dimitriadis, presented as Nagano and Valian 2011) and, thanks to Python being a general-purpose programming language, this allows virtually anything to be done with the parsed data structure. There is, however, one crucial criterion if one would like to use NLTK to handle CHAT transcripts: NLTK currently requires the XML version (a mark-up schema devised by the CHILDES team) of the CHAT transcripts. Such a requirement adds an additional layer of work and effort, thereby increasing the chance of introducing errors in the workflow. Although CHILDES does provide tools that convert CHAT transcripts into their specified XML format, this requires that the CHAT format specifications and the associated tools (all updated from time to time) be mutually compatible, which could be overlooked in actual use and create confusion. Moreover, it is clear that human researchers work most comfortably and conveniently with CHAT transcripts directly, not with the derived XML version with rich mark-up language that is not intended to be handled by humans.

Given this background, there is a need for a general tool that parses CHAT transcripts and allows researchers to write their own scripts and programs to interact with the parsed data structure. In this section, we introduce the Python library PyLangAcq for exactly these purposes.[1] Our choice of programming language is due to the widespread use of Python in computational linguistics and natural language processing. PyLangAcq makes it possible that the great variety of machine learning as well as other computational and statistical tools available via Python can be used to model any phenomena of interest with respect to CHAT datasets. As the CHAT format is used for speech transcriptions more generally, PyLangAcq will be useful for researchers of many other linguistically related fields.

PyLangAcq is ever expanding and evolving, with its official detailed documentation hosted online and regularly updated (`http://pylangacq.org/`). At the time of writing, PyLangAcq is fully operational for parsing CHAT transcripts and extracting information of

1. `https://pylangacq.org/`

interest, including but not limited to the following:

- participants (e.g., `CHI` (target child), `MOT` (mother)) and their demographic information
- age (of the target child, most typically)
- transcriptions in various data structures
- word frequency information and n-grams
- word search and concordance
- dependency graphs (based on `%gra` tiers)
- standard language development measures such as type-token ratio (TTR), mean length of utterance (MLU), and index of productive syntax (IPSyn)

The reader is directed to the online documentation of the library for any of these items and more. They are the building blocks of advanced modules and functions currently being developed and added to the library.

In the rest of this chapter, we illustrate the use of PyLangAcq for measuring the mean length of utterance in morphemes (MLUm; section 5.2.1), studying bilingualism (section 5.2.2), and exploring phonological development (section 5.2.3).

5.2.1 The mean length of utterance in morphemes (MLUm)

We illustrate how the mean length of utterance in morphemes (MLUm) can be computed by PyLangAcq for some given CHILDES dataset in language acquisition research. Our example uses Eve's data from the Brown portion (Brown, 1973) of CHILDES. As MLUm is often used as a measure of language development, we may ask if MLUm is correlated with age in Eve's data. Figure 5.1 shows the results:

Filename	Age (months)	MLUm
`eve01.cha`	18	2.267
`eve02.cha`	18	2.449
`eve03.cha`	19	2.763
`eve04.cha`	19	2.576
`eve05.cha`	20	2.859
`eve06.cha`	21	3.1
`eve07.cha`	21	3.1
`eve08.cha`	21	3.3
`eve09.cha`	22	3.8
`eve10.cha`	22	3.7
`eve11.cha`	23	3.8
`eve12.cha`	23	4.1
`eve13.cha`	24	4.2
`eve14.cha`	24	3.9
`eve15.cha`	25	4.450
`eve16.cha`	25	4.424
`eve17.cha`	26	4.466
`eve18.cha`	26	4.288
`eve19.cha`	27	4.348
`eve20.cha`	27	3.163

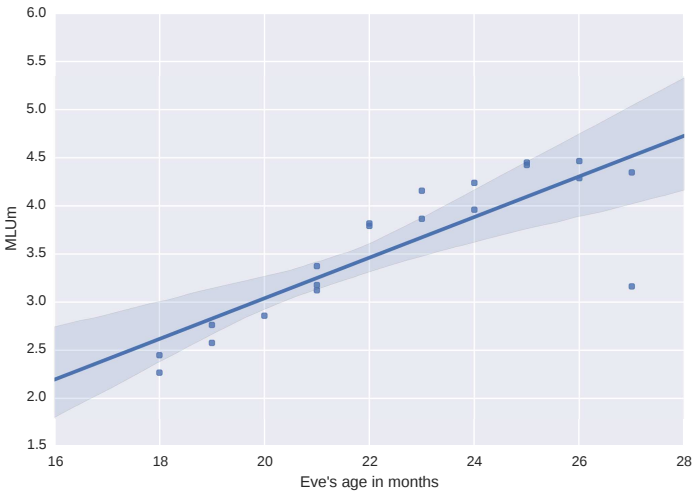


Figure 5.1: Eve's MLUm at different ages (Pearson's $r = 0.84, p < 0.001$)

The results show that Eve's age is significantly (and positively) correlated with MLUm (Pearson's $r = 0.84, p < 0.001$).

The code for MLUm computation illustrated above is available online. The computation of MLUm is performed entirely in Python from reading the dataset all the way to data

analysis and visualization, combining PyLangAcq with other Python libraries and tools.[2]

5.2.2 Language dominance measured by MLUw

As the mean length of utterance is commonly used as a measure of language development, it is used in a wide variety of research topics in language acquisition. We illustrate how PyLangAcq can be used in research on bilingualism, specifically in the area of bilingual first language acquisition.

An important aspect of bilingualism concerns how various factors might contribute to the developmental trajectories of different languages spoken by a bilingual speaker. Essential in this research area is a reliable means of measuring language dominance, for whether (and by how much) a bilingual speaker is more competent in one language than in another. Here, we use PyLangAcq to replicate some of the results of—and possibly provide new insights for—Yip and Matthews (2007) for language dominance.

We focus on the datasets from the three siblings Timmy (eldest), Sophie, and Alicia (youngest) from the "YipMatthews" corpus in the "Biling" section of CHILDES. Born and raised in Hong Kong, they are Cantonese-English bilinguals whose mother is a native speaker of Cantonese and whose father is a native speaker of English. Following Yip and Matthews (2007, 73-81), we compare patterns of language dominance (measured by MLUw, the mean length of utterance in words) of these three children acquiring Cantonese and English simultaneously:[3]

2. The complete code is here: `https://github.com/jacksonllee/pylangacq/blob/main/docs/papers/tech-report-2016.py`
Other libraries and tools of Python (van Rossum and Drake Jr, 1995) we have used are IPython Notebook (Pérez and Granger, 2007), SciPy (Jones et al., 2001–), pandas (McKinney, 2010), and Seaborn (Waskom, 2015) (built upon matplotlib (Hunter, 2007)).

3. The complete code is also available online. See footnote 2.

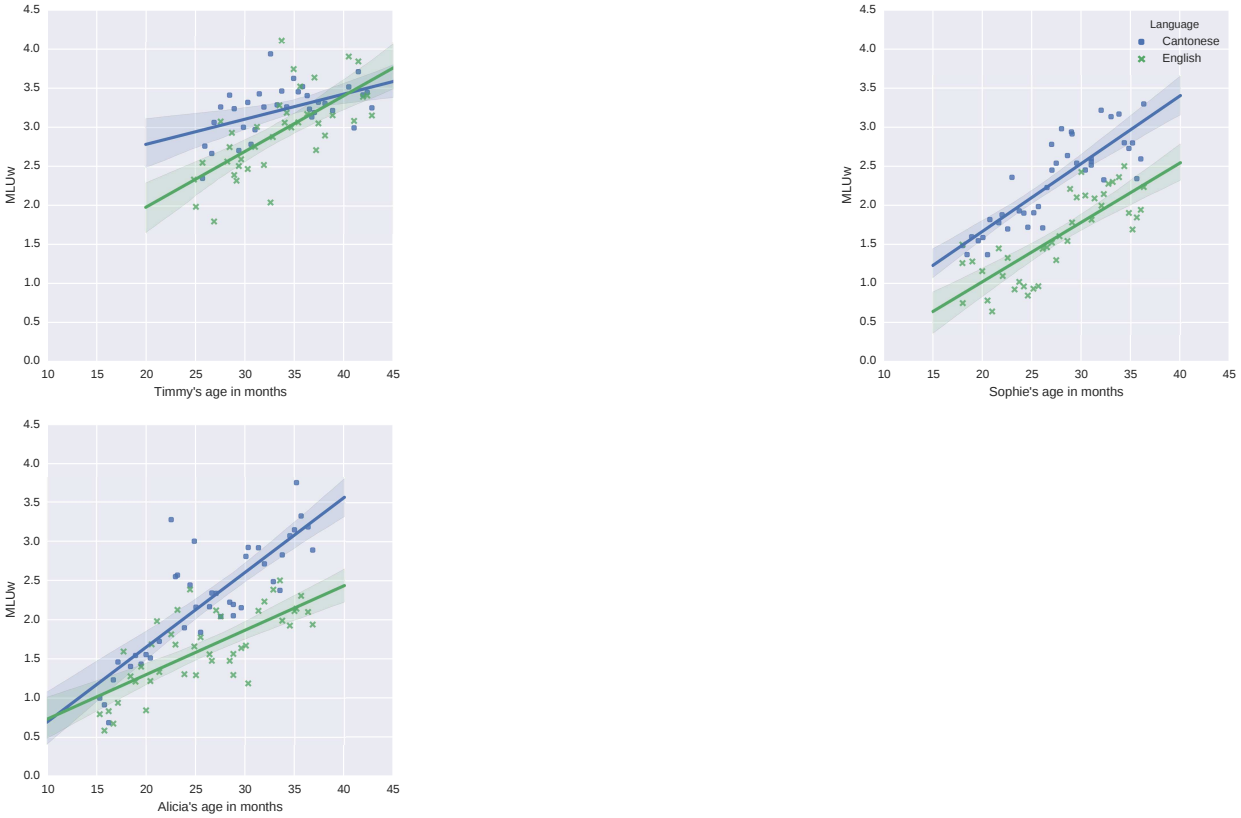


Figure 5.2: MLUw of Timmy, Sophie, and Alicia from CHILDES YipMatthews

For the purposes of comparison, the three plots in Figure 5.2 are produced with the same axes and ranges of values. For each language of each child, the best fit lines showing the overall trajectory (with shaded error regions at the 95% confidence interval) are also given.

Based on Figure 5.2, several observations are borne out, which are also discussed by Yip and Matthews. The three children appear to exhibit Cantonese dominance in general, but with interesting differences. Timmy, the eldest sibling, shows higher competence in Cantonese early on, but his English caught up quickly during the period of study. For Sophie and Alicia, Yip and Matthews (2007, 77) point out that they show a consistent pattern of Cantonese dominance. This seems to be the case as shown in Figure 5.2, although Alicia shows the pattern of relatively increasing preference of Cantonese, whereas Sophie's competence in Cantonese and English matures in a more or less comparable rate. And yet Sophie and Alicia's patterns contrast sharply with Timmy's, whose preferential growth of

English competence during the period of study is unobserved in his sisters. It is interesting to see how divergent bilingual development can be – even within the same family. While these finer-grained observations are possibly tangential to the particular research that Yip and Matthews (2007) focus on, the fact that more detailed statistical analyses and data visualization are available in a purely Python environment incorporating PyLangAcq shows that PyLangAcq can facilitate language acquisition research for large datasets and more sophisticated computational and statistical analysis.

5.2.3 Phonological development

PyLangAcq also facilitates research by use in conjunction with other Python tools developed particularly for linguistics. Continuing with Cantonese, one of the languages exemplified above, we use PyCantonese (Lee, 2015b), a Python library for Cantonese linguistic research. In the following, we briefly explore phonological development – child tone production in particular; Cantonese is a tone language with six tones. In this example, PyLangAcq handles the CHILDES Cantonese monolingual child development data from Lee and Wong (1998), and PyCantonese parses Cantonese romanization for extracting tone information.

We use the data from the child MHZ. There are altogether 16 CHAT data files, with the age range of 24.5-32.2 months. As a first step for future work, we briefly explore the distribution of tones produced by MHZ. For each file, we count the number of times a particular tone is produced by MHZ. The results are presented in the following heatmap:[4]

4. The complete code for this part is also available online. See footnote 2.

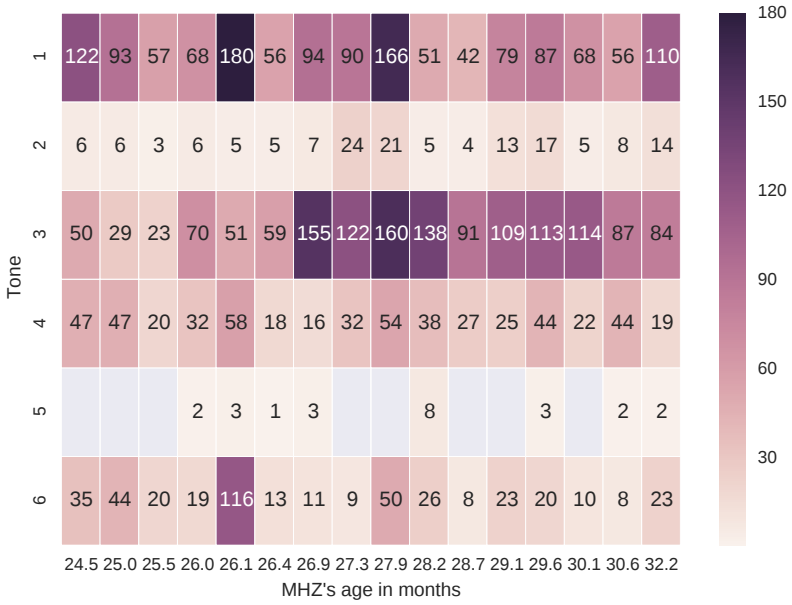


Figure 5.3: MHZ's tone production

With the aid of heatmap visualization in Figure 5.3, we can see that the level tones (tones 1, 3, 6) and the low-falling/level tone 4 appear to be most frequently produced by the child. The observation that the level tones are empirically more frequent than contour tones is similar to findings with regard to adult Cantonese corpus studies (Leung et al., 2004). Possible further research which could be performed using PyLangAcq includes comparing children's speech, as shown above, to child-directed speech, and modeling the development of tone production distribution over time.

5.3 Modeling human morphological acquisition

The previous chapter on paradigm induction and alignment introduced *Linguistica 5* for learning morphological paradigms from raw text. A powerful feature of *Linguistica 5* is that it is a Python library by design and is therefore callable in other Python-based programs. This is significant, because it is now possible to run the Linguistica algorithms dynamically. With PyLangAcq available, we can ask how it can be used to model human morphological

learning using child-directed speech data. An important criterion is that for the model to be cognitively plausible, it has to simulate the incremental nature of the input data. This means that the Linguistica algorithm for morphological learning must be called and applied flexibly over some growing data.

Concretely, we tested *Linguistica 5* for its ability to model morphological acquisition using Eve's data in the Brown portion Brown (1973) of the CHILDES database MacWhinney (2000), an idea sketched in Lee (2015a). The child-directed speech (CDS) at different ages of the target child in the data was extracted by the PyLangAcq library Lee et al. (2016) and fed into *Linguistica 5*. Table 5.1 shows the results of morphological signature induction from growing word types up to the ages of 18, 21, and 24 months, respectively.

Age	# word types	Induced signatures
18 mths	610	{'s Ø}{Ø s}
21 mths	1,246	{'s Ø}{Ø s}{Ø ing}{ll s}
24 mths	1,601	{'s Ø}{Ø s}{Ø ing}{ll s}{'s Ø s}

Table 5.1: Morphological signatures from CDS to Eve

The classic study of first language acquisition by Brown (1973) reports that the first three morphological patterns acquired by English-speaking children are the third-person singular inflection {Ø, s}, the possessive {'s, Ø}, and the progressive {Ø, ing}. Table 5.1 shows these are patterns that *Linguistica 5* successfully discovers in Eve's child-directed speech. Other induced signatures are {ll, s} (as in *she'll-she's*) and {'s, Ø, s}, a more complex pattern found when more data becomes available to the learner. The results for modeling language acquisition here contrast sharply with those from the Brown corpus in section 4.3.1, for the much larger amount of input data, and the qualitatively different results, in the latter case. But of particular interest is the *incremental* nature of learning in the former case. The fact that *Linguistica 5* is a Python library makes it possible to devise tools embedding it for multiple learning iterations run automatically.

APPENDIX: STEM EXTRACTION RESULTS FOR ENGLISH

substring: 12005
subsequence: 11619
submultiset: 14329

---

be
be, is, was, been, being

substring (cost: 80)

subsequence (cost: 80)

submultiset (cost: 80)

---

have
have, has, had, had, having

substring (cost: 80)
ha␣␣ve ␣␣s ␣␣d ␣␣d ␣␣ving
~ve ~s ~d ~d ~ving
subsequence (cost: 80)
ha␣␣ve ␣␣s ␣␣d ␣␣d ␣␣ving
~ve ~s ~d ~d ~ving
submultiset (cost: 80)
ah␣␣ve ␣␣s ␣␣d ␣␣d ␣␣ving
~ve ~s ~d ~d ~ving

---

do
do, does, did, done, doing

substring (cost: 111)
d␣o ␣oes ␣id, di␣ ␣one ␣oing
~o ~oes ~id, di~ ~one ~oing

subsequence (cost: 111)
 d ␣o ␣oes ␣id, di␣ ␣one ␣oing
 ∼o ∼oes ∼id, di∼ ∼one ∼oing
submultiset (cost: 111)
 d ␣o ␣oes ␣id, di␣ ␣one ␣oing
 ∼o ∼oes ∼id, di∼ ∼one ∼oing

---

say
say, says, said, said, saying

substring (cost: 90)
 sa ␣␣y ␣␣ys ␣␣id ␣␣id ␣␣ying
 ∼y ∼ys ∼id ∼id ∼ying
subsequence (cost: 90)
 sa ␣␣y ␣␣ys ␣␣id ␣␣id ␣␣ying
 ∼y ∼ys ∼id ∼id ∼ying
submultiset (cost: 111)
 as ␣␣y ␣␣ys, s␣y␣ ␣␣id ␣␣id ␣␣ying
 ∼y ∼ys, s∼y∼ ∼id ∼id ∼ying

---

go
go, goes, went, gone, going

substring (cost: 95)

subsequence (cost: 95)

submultiset (cost: 95)

---

get
get, gets, got, gotten, getting

substring (cost: 344)

g	␣et	␣ets	␣ot	␣otten	␣etting, gettin␣
	∼et	∼ets	∼ot	∼otten	∼etting, gettin∼
t	ge␣	ge␣s	go␣	go␣ten, got␣en	ge␣ting, get␣ing
	ge∼	ge∼s	go∼	go∼ten, got∼en	ge∼ting, get∼ing

subsequence (cost: 192)

gt	␣e␣	␣e␣s	␣o␣	␣o␣ten, ␣ot␣en	␣e␣ting, ␣et␣ing
	∼e∼	∼e∼s	∼o∼	∼o∼ten, ∼ot∼en	∼e∼ting, ∼et∼ing

submultiset (cost: 263)

gt	␣e␣	␣e␣s	␣o␣	␣o␣ten, ␣ot␣en	␣e␣ting, ␣et␣ing, ge␣tin␣, get␣in␣
	∼e∼	∼e∼s	∼o∼	∼o∼ten, ∼ot∼en	∼e∼ting, ∼et∼ing, ge∼tin∼, get∼in∼

---

know

know, knows, knew, known, knowing

substring (cost: 110)

kn	␣␣ow	␣␣ows	␣␣ew	␣␣own	␣␣owing
	∼ow	∼ows	∼ew	∼own	∼owing

subsequence (cost: 115)

knw	␣␣o␣	␣␣o␣s	␣␣e␣	␣␣o␣n	␣␣o␣ing
	∼o∼	∼o∼s	∼e∼	∼o∼n	∼o∼ing

submultiset (cost: 172)

knw	␣␣o␣	␣␣o␣s	␣␣e␣	␣␣o␣n, ␣no␣␣	␣␣o␣ing, ␣no␣i␣g
	∼o∼	∼o∼s	∼e∼	∼no∼, ∼o∼n	∼no∼i∼g, ∼o∼ing

---

make

make, makes, made, made, making

substring (cost: 100)

ma	␣␣ke	␣␣kes	␣␣de	␣␣de	␣␣king
	∼ke	∼kes	∼de	∼de	∼king

subsequence (cost: 100)

ma	␣␣ke	␣␣kes	␣␣de	␣␣de	␣␣king
	∼ke	∼kes	∼de	∼de	∼king

submultiset (cost: 100)

am	␣␣ke	␣␣kes	␣␣de	␣␣de	␣␣king
	∼ke	∼kes	∼de	∼de	∼king

---

think

think, thinks, thought, thought, thinking

substring (cost: 150)

th	␣␣ink	␣␣inks	␣␣ought	␣␣ought	␣␣inking
	∼ink	∼inks	∼ought	∼ought	∼inking

subsequence (cost: 222)

th	␣␣ink	␣␣inks	␣␣ought, ␣houg␣t	␣␣ought, ␣houg␣t	␣␣inking
	∼ink	∼inks	∼houg∼t, ∼ought	∼houg∼t, ∼ought	∼inking

submultiset (cost: 354)

ht	␣␣ink	␣␣inks	␣␣ought, ␣houg␣t, t␣ough␣, thoug␣␣		
	∼ink	∼inks	∼houg∼t, ∼ought, t∼ough∼, thoug∼		

---

take

take, takes, took, taken, taking

substring (cost: 251)

k	ta␣e	ta␣es	too␣	ta␣en	ta␣ing
	ta∼e	ta∼es	too∼	ta∼en	ta∼ing
t	␣ake	␣akes	␣ook	␣aken	␣aking
	∼ake	∼akes	∼ook	∼aken	∼aking

subsequence (cost: 130)

tk	␣a␣e	␣a␣es	␣oo␣	␣a␣en	␣a␣ing
	∼a∼e	∼a∼es	∼oo∼	∼a∼en	∼a∼ing

submultiset (cost: 130)

kt	␣a␣e	␣a␣es	␣oo␣	␣a␣en	␣a␣ing
	∼a∼e	∼a∼es	∼oo∼	∼a∼en	∼a∼ing

---

see

see, sees, saw, seen, seeing

substring (cost: 126)

s	␣ee	␣ees, see␣	␣aw	␣een	␣eeing
	∼ee	∼ees, see∼	∼aw	∼een	∼eeing

subsequence (cost: 126)

s	␣ee	␣ees, see␣	␣aw	␣een	␣eeing
	∼ee	∼ees, see∼	∼aw	∼een	∼eeing

submultiset (cost: 126)

s	␣ee	␣ees, see␣	␣aw	␣een	␣eeing
	∼ee	∼ees, see∼	∼aw	∼een	∼eeing

---

come

come, comes, came, come, coming

substring (cost: 241)

c	␣ome	␣omes	␣ame	␣ome	␣oming
	∼ome	∼omes	∼ame	∼ome	∼oming
m	co␣e	co␣es	ca␣e	co␣e	co␣ing
	co∼e	co∼es	ca∼e	co∼e	co∼ing

subsequence (cost: 125)

cm	␣o␣e	␣o␣es	␣a␣e	␣o␣e	␣o␣ing
	∼o∼e	∼o∼es	∼a∼e	∼o∼e	∼o∼ing

submultiset (cost: 125)

cm	␣o␣e	␣o␣es	␣a␣e	␣o␣e	␣o␣ing
	∼o∼e	∼o∼es	∼a∼e	∼o∼e	∼o∼ing

---

want

want, wants, wanted, wanted, wanting

substring (cost: 85)

want	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

want	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 116)

antw	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing, ␣␣n␣i␣g
	∼	∼s	∼ed	∼ed	∼ing, ∼n∼i∼g

---

look

look, looks, looked, looked, looking

substring (cost: 85)

look	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

look	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 85)

kloo	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

---

use

use, uses, used, used, using

substring (cost: 85)

us	␣␣e	␣␣es	␣␣ed	␣␣ed	␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

subsequence (cost: 106)

us	␣␣e	␣␣es, ␣se␣	␣␣ed	␣␣ed	␣␣ing
	∼e	∼es, ∼se∼	∼ed	∼ed	∼ing

submultiset (cost: 106)

su	␣␣e	␣␣es, ␣se␣	␣␣ed	␣␣ed	␣␣ing
	∼e	∼es, ∼se∼	∼ed	∼ed	∼ing

---

find

find, finds, found, found, finding

substring (cost: 115)

nd	fi␣␣	fi␣␣s	fou␣␣	fou␣␣	fi␣␣ing
	fi∼	fi∼s	fou∼	fou∼	fi∼ing

subsequence (cost: 120)

fnd	␣i␣␣	␣i␣␣s	␣ou␣␣	␣ou␣␣	␣i␣␣ing
	∼i∼	∼i∼s	∼ou∼	∼ou∼	∼i∼ing

submultiset (cost: 156)

dfn	␣i␣␣	␣i␣␣s	␣ou␣␣	␣ou␣␣	␣i␣␣ing, ␣in␣i␣g
	∼i∼	∼i∼s	∼ou∼	∼ou∼	∼i∼ing, ∼in∼i∼g

---

tell

tell, tells, told, told, telling

substring (cost: 334)

l	te␣l, tel␣	te␣ls, tel␣s	to␣d	to␣d	te␣ling, tel␣ing
	te∼l, tel∼	te∼ls, tel∼s	to∼d	to∼d	te∼ling, tel∼ing
t	␣ell	␣ells	␣old	␣old	␣elling
	∼ell	∼ells	∼old	∼old	∼elling

subsequence (cost: 213)

tl	␣e␣l, ␣el␣	␣e␣ls, ␣el␣s	␣o␣d	␣o␣d	␣e␣ling, ␣el␣ing
	∼e∼l, ∼el∼	∼e∼ls, ∼el∼s	∼o∼d	∼o∼d	∼e∼ling, ∼el∼ing

submultiset (cost: 213)

lt	␣e␣l, ␣el␣	␣e␣ls, ␣el␣s	␣o␣d	␣o␣d	␣e␣ling, ␣el␣ing
	∼e∼l, ∼el∼	∼e∼ls, ∼el∼s	∼o∼d	∼o∼d	∼e∼ling, ∼el∼ing

---

give

give, gives, gave, given, giving

substring (cost: 282)

g	␣ive	␣ives	␣ave	␣iven	␣iving, givin␣
	∼ive	∼ives	∼ave	∼iven	∼iving, givin∼
v	gi␣e	gi␣es	ga␣e	gi␣en	gi␣ing
	gi∼e	gi∼es	ga∼e	gi∼en	gi∼ing

subsequence (cost: 130)

gv	␣i␣e	␣i␣es	␣a␣e	␣i␣en	␣i␣ing
	∼i∼e	∼i∼es	∼a∼e	∼i∼en	∼i∼ing

submultiset (cost: 161)

gv	␣i␣e	␣i␣es	␣a␣e	␣i␣en	␣i␣ing, gi␣in␣
	∼i∼e	∼i∼es	∼a∼e	∼i∼en	∼i∼ing, gi∼in∼

---

work

work, works, worked, worked, working

substring (cost: 85)

work	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

work	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 85)

korw	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

---

call

call, calls, called, called, calling

substring (cost: 85)

call	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

call	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 85)

acll	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

---

try

try, tries, tried, tried, trying

substring (cost: 105)
 tr ␣␣y ␣␣ies ␣␣ied ␣␣ied ␣␣ying
 ∼y ∼ies ∼ied ∼ied ∼ying
subsequence (cost: 105)
 tr ␣␣y ␣␣ies ␣␣ied ␣␣ied ␣␣ying
 ∼y ∼ies ∼ied ∼ied ∼ying
submultiset (cost: 105)
 rt ␣␣y ␣␣ies ␣␣ied ␣␣ied ␣␣ying
 ∼y ∼ies ∼ied ∼ied ∼ying

---

ask
ask, asks, asked, asked, asking

substring (cost: 80)
 ask ␣␣␣ ␣␣␣s ␣␣␣ed ␣␣␣ed ␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 80)
 ask ␣␣␣ ␣␣␣s ␣␣␣ed ␣␣␣ed ␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
submultiset (cost: 96)
 aks ␣␣␣ ␣␣␣s, ␣s␣␣ ␣␣␣ed ␣␣␣ed ␣␣␣ing
 ∼ ∼s, ∼s∼ ∼ed ∼ed ∼ing

---

need
need, needs, needed, needed, needing

substring (cost: 85)
 need ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 219)
 need ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed, ␣␣␣de␣, ␣␣ed␣␣, ␣e␣d␣␣ ␣␣␣␣ed, ␣␣␣de␣, ␣␣ed␣␣, ␣e␣d␣␣ ␣␣␣␣ing
 ∼ ∼s ∼de∼, ∼e∼d∼, ∼ed, ∼ed∼ ∼de∼, ∼e∼d∼, ∼ed, ∼ed∼ ∼ing
submultiset (cost: 287)

deen	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed, ␣␣␣de␣, ␣␣e␣␣d, ␣␣ed␣␣, ␣e␣␣␣d, ␣e␣d␣␣
	∼	∼s	∼de∼, ∼e∼d, ∼e∼d∼, ∼ed, ∼ed∼

feel
feel, feels, felt, felt, feeling

substring (cost: 211)

el	fe␣␣	fe␣␣s	f␣␣t	f␣␣t	fe␣␣ing
	fe∼	fe∼s	f∼t	f∼t	fe∼ing
fe	␣␣el	␣␣els	␣␣lt	␣␣lt	␣␣eling
	∼el	∼els	∼lt	∼lt	∼eling

subsequence (cost: 100)

fel	␣␣e␣, ␣e␣␣	␣␣e␣s, ␣e␣␣s	␣␣␣t	␣␣␣t	␣␣e␣ing, ␣e␣␣ing
	∼e∼	∼e∼s	∼t	∼t	∼e∼ing

submultiset (cost: 100)

efl	␣␣e␣, ␣e␣␣	␣␣e␣s, ␣e␣␣s	␣␣␣t	␣␣␣t	␣␣e␣ing, ␣e␣␣ing
	∼e∼	∼e∼s	∼t	∼t	∼e∼ing

become
become, becomes, became, become, becoming

substring (cost: 130)

bec	␣␣␣ome	␣␣␣omes	␣␣␣ame	␣␣␣ome	␣␣␣oming
	∼ome	∼omes	∼ame	∼ome	∼oming

subsequence (cost: 135)

becm	␣␣␣o␣e	␣␣␣o␣es	␣␣␣a␣e	␣␣␣o␣e	␣␣␣o␣ing
	∼o∼e	∼o∼es	∼a∼e	∼o∼e	∼o∼ing

submultiset (cost: 244)

bcem	␣␣␣o␣e, ␣e␣o␣␣	␣␣␣o␣es, ␣e␣o␣␣s	␣␣␣a␣e, ␣e␣a␣␣	␣␣␣o␣e, ␣e␣o␣␣	␣␣␣o␣ing
	∼e∼o∼, ∼o∼e	∼e∼o∼s, ∼o∼es	∼a∼e, ∼e∼a∼	∼e∼o∼, ∼o∼e	∼o∼ing

mean
mean, means, meant, meant, meaning

substring (cost: 75)
 mean ␣␣␣␣ ␣␣␣␣s ␣␣␣␣t ␣␣␣␣t ␣␣␣␣ing
 ∼ ∼s ∼t ∼t ∼ing
subsequence (cost: 101)
 mean ␣␣␣␣ ␣␣␣␣s ␣␣␣␣t ␣␣␣␣t ␣␣␣␣ing, ␣␣␣ni␣g
 ∼ ∼s ∼t ∼t ∼ing, ∼ni∼g
submultiset (cost: 101)
 aemn ␣␣␣␣ ␣␣␣␣s ␣␣␣␣t ␣␣␣␣t ␣␣␣␣ing, ␣␣␣ni␣g
 ∼ ∼s ∼t ∼t ∼ing, ∼ni∼g

---

leave
leave, leaves, left, left, leaving

substring (cost: 115)
 le ␣␣ave ␣␣aves ␣␣ft ␣␣ft ␣␣aving
 ∼ave ∼aves ∼ft ∼ft ∼aving
subsequence (cost: 172)
 le ␣␣ave, ␣eav␣ ␣␣aves, ␣eav␣s ␣␣ft ␣␣ft ␣␣aving
 ∼ave, ∼eav∼ ∼aves, ∼eav∼s ∼ft ∼ft ∼aving
submultiset (cost: 172)
 el ␣␣ave, ␣eav␣ ␣␣aves, ␣eav␣s ␣␣ft ␣␣ft ␣␣aving
 ∼ave, ∼eav∼ ∼aves, ∼eav∼s ∼ft ∼ft ∼aving

---

let
let, lets, let, let, letting

substring (cost: 65)
 let ␣␣␣ ␣␣␣s ␣␣␣ ␣␣␣ ␣␣␣ting
 ∼ ∼s ∼ ∼ ∼ting
subsequence (cost: 96)
 let ␣␣␣ ␣␣␣s ␣␣␣ ␣␣␣ ␣␣␣ting, ␣␣t␣ing
 ∼ ∼s ∼ ∼ ∼t∼ing, ∼ting
submultiset (cost: 96)

elt	␣␣␣	␣␣␣s	␣␣␣	␣␣␣	␣␣␣ting, ␣␣t␣ing
	∼	∼s	∼	∼	∼t∼ing, ∼ting

---

put
put, puts, put, put, putting

substring (cost: 65)

put	␣␣␣	␣␣␣s	␣␣␣	␣␣␣	␣␣␣ting
	∼	∼s	∼	∼	∼ting

subsequence (cost: 96)

put	␣␣␣	␣␣␣s	␣␣␣	␣␣␣	␣␣␣ting, ␣␣t␣ing
	∼	∼s	∼	∼	∼t∼ing, ∼ting

submultiset (cost: 96)

ptu	␣␣␣	␣␣␣s	␣␣␣	␣␣␣	␣␣␣ting, ␣␣t␣ing
	∼	∼s	∼	∼	∼t∼ing, ∼ting

---

keep
keep, keeps, kept, kept, keeping

substring (cost: 211)

ep	ke␣␣	ke␣␣s	k␣␣t	k␣␣t	ke␣␣ing
	ke∼	ke∼s	k∼t	k∼t	ke∼ing
ke	␣␣ep	␣␣eps	␣␣pt	␣␣pt	␣␣eping
	∼ep	∼eps	∼pt	∼pt	∼eping

subsequence (cost: 100)

kep	␣␣e␣, ␣e␣␣	␣␣e␣s, ␣e␣␣s	␣␣␣t	␣␣␣t	␣␣e␣ing, ␣e␣␣ing
	∼e∼	∼e∼s	∼t	∼t	∼e∼ing

submultiset (cost: 100)

ekp	␣␣e␣, ␣e␣␣	␣␣e␣s, ␣e␣␣s	␣␣␣t	␣␣␣t	␣␣e␣ing, ␣e␣␣ing
	∼e∼	∼e∼s	∼t	∼t	∼e∼ing

---

talk
talk, talks, talked, talked, talking

substring (cost: 85)
 talk ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 85)
 talk ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
submultiset (cost: 85)
 aklt ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing

---

turn
turn, turns, turned, turned, turning

substring (cost: 85)
 turn ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 111)
 turn ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing, ␣␣␣ni␣g
 ∼ ∼s ∼ed ∼ed ∼ing, ∼ni∼g
submultiset (cost: 111)
 nrtu ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing, ␣␣␣ni␣g
 ∼ ∼s ∼ed ∼ed ∼ing, ∼ni∼g

---

seem
seem, seems, seemed, seemed, seeming

substring (cost: 85)
 seem ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 85)
 seem ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
submultiset (cost: 138)

eems	␣␣␣␣	␣␣␣␣s, s␣␣␣␣	␣␣␣␣ed, ␣␣e␣␣d, ␣e␣␣␣d	␣␣␣␣ed, ␣␣e␣␣d, ␣e␣␣␣d	␣␣␣␣ing
	∼	∼s, s∼	∼e∼d, ∼ed	∼e∼d, ∼ed	∼ing

begin
begin, begins, began, begun, beginning

substring (cost: 115)

beg	␣␣␣in	␣␣␣ins	␣␣␣an	␣␣␣un	␣␣␣inning
	∼in	∼ins	∼an	∼un	∼inning

subsequence (cost: 192)

begn	␣␣␣i␣	␣␣␣i␣s	␣␣␣a␣	␣␣␣u␣	␣␣␣i␣ning, ␣␣␣in␣ing, ␣␣␣inni␣g
	∼i∼	∼i∼s	∼a∼	∼u∼	∼i∼ning, ∼in∼ing, ∼inni∼g

submultiset (cost: 308)

begn	␣␣␣i␣	␣␣␣i␣s	␣␣␣a␣	␣␣␣u␣	
	∼i∼	∼i∼s	∼a∼	∼u∼	

help
help, helps, helped, helped, helping

substring (cost: 85)

help	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

help	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 127)

ehlp	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed, ␣e␣␣␣d	␣␣␣␣ed, ␣e␣␣␣d	␣␣␣␣ing
	∼	∼s	∼e∼d, ∼ed	∼e∼d, ∼ed	∼ing

start
start, starts, started, started, starting

substring (cost: 90)

start	␣␣␣␣␣	␣␣␣␣␣s	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 90)
 start ␣␣␣␣␣ ␣␣␣␣␣s ␣␣␣␣␣ed ␣␣␣␣␣ed ␣␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
submultiset (cost: 101)
 arstt ␣␣␣␣␣ ␣␣␣␣␣s, s␣␣␣␣␣ ␣␣␣␣␣ed ␣␣␣␣␣ed ␣␣␣␣␣ing
 ∼ ∼s, s∼ ∼ed ∼ed ∼ing

---

show
show, shows, showed, showed, showing

substring (cost: 85)
 show ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 85)
 show ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
submultiset (cost: 96)
 hosw ␣␣␣␣ ␣␣␣␣s, s␣␣␣␣ ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s, s∼ ∼ed ∼ed ∼ing

---

hear
hear, hears, heard, heard, hearing

substring (cost: 75)
 hear ␣␣␣␣ ␣␣␣␣s ␣␣␣␣d ␣␣␣␣d ␣␣␣␣ing
 ∼ ∼s ∼d ∼d ∼ing
subsequence (cost: 75)
 hear ␣␣␣␣ ␣␣␣␣s ␣␣␣␣d ␣␣␣␣d ␣␣␣␣ing
 ∼ ∼s ∼d ∼d ∼ing
submultiset (cost: 75)
 aehr ␣␣␣␣ ␣␣␣␣s ␣␣␣␣d ␣␣␣␣d ␣␣␣␣ing
 ∼ ∼s ∼d ∼d ∼ing

---

play

play, plays, played, played, playing

substring (cost: 85)

play	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

play	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 85)

alpy	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

---

run

run, runs, ran, run, running

substring (cost: 283)

n	ru␣	ru␣s	ra␣	ru␣	ru␣ning, run␣ing, runni␣g
	ru∼	ru∼s	ra∼	ru∼	ru∼ning, run∼ing, runni∼g
r	␣un	␣uns	␣an	␣un	␣unning
	∼un	∼uns	∼an	∼un	∼unning

subsequence (cost: 182)

rn	␣u␣	␣u␣s	␣a␣	␣u␣	␣u␣ning, ␣un␣ing, ␣unni␣g
	∼u∼	∼u∼s	∼a∼	∼u∼	∼u∼ning, ∼un∼ing, ∼unni∼g

submultiset (cost: 182)

nr	␣u␣	␣u␣s	␣a␣	␣u␣	␣u␣ning, ␣un␣ing, ␣unni␣g
	∼u∼	∼u∼s	∼a∼	∼u∼	∼u∼ning, ∼un∼ing, ∼unni∼g

---

happen

happen, happens, happened, happened, happening

substring (cost: 95)

happen	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 121)

happen	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing, ␣␣␣␣␣ni␣g
	∼	∼s	∼ed	∼ed	∼ing, ∼ni∼g

submultiset (cost: 163)

aehnpp	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed, ␣␣␣␣e␣␣d	␣␣␣␣␣␣ed, ␣␣␣␣e␣␣d	␣␣␣␣␣␣ing, ␣␣␣␣␣ni␣g
	∼	∼s	∼e∼d, ∼ed	∼e∼d, ∼ed	∼ing, ∼ni∼g

---

like

like, likes, liked, liked, liking

substring (cost: 90)

lik	␣␣␣e	␣␣␣es	␣␣␣ed	␣␣␣ed	␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

subsequence (cost: 90)

lik	␣␣␣e	␣␣␣es	␣␣␣ed	␣␣␣ed	␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

submultiset (cost: 116)

ikl	␣␣␣e	␣␣␣es	␣␣␣ed	␣␣␣ed	␣␣␣ing, ␣i␣␣ng
	∼e	∼es	∼ed	∼ed	∼i∼ng, ∼ing

---

move

move, moves, moved, moved, moving

substring (cost: 90)

mov	␣␣␣e	␣␣␣es	␣␣␣ed	␣␣␣ed	␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

subsequence (cost: 90)

mov	␣␣␣e	␣␣␣es	␣␣␣ed	␣␣␣ed	␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

submultiset (cost: 90)

mov	␣␣␣e	␣␣␣es	␣␣␣ed	␣␣␣ed	␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

---

believe

believe, believes, believed, believed, believing

substring (cost: 105)
 believ ␣␣␣␣␣␣e ␣␣␣␣␣␣es ␣␣␣␣␣␣ed ␣␣␣␣␣␣ed ␣␣␣␣␣␣ing
 ∼e ∼es ∼ed ∼ed ∼ing
subsequence (cost: 105)
 believ ␣␣␣␣␣␣e ␣␣␣␣␣␣es ␣␣␣␣␣␣ed ␣␣␣␣␣␣ed ␣␣␣␣␣␣ing
 ∼e ∼es ∼ed ∼ed ∼ing
submultiset (cost: 210)
 beeilv ␣␣␣␣␣␣e, ␣␣␣␣e␣␣, ␣e␣␣␣␣ ␣␣␣␣␣␣es, ␣␣␣␣e␣␣s, ␣e␣␣␣␣␣s
 ∼e, ∼e∼ ∼e∼s, ∼es

---

hold
hold, holds, held, held, holding

substring (cost: 105)
 ld ho␣␣ ho␣␣s he␣␣ he␣␣ ho␣␣ing
 ho∼ ho∼s he∼ he∼ ho∼ing
subsequence (cost: 110)
 hld ␣o␣␣ ␣o␣␣s ␣e␣␣ ␣e␣␣ ␣o␣␣ing
 ∼o∼ ∼o∼s ∼e∼ ∼e∼ ∼o∼ing
submultiset (cost: 110)
 dhl ␣o␣␣ ␣o␣␣s ␣e␣␣ ␣e␣␣ ␣o␣␣ing
 ∼o∼ ∼o∼s ∼e∼ ∼e∼ ∼o∼ing

---

live
live, lives, lived, lived, living

substring (cost: 90)
 liv ␣␣␣e ␣␣␣es ␣␣␣ed ␣␣␣ed ␣␣␣ing
 ∼e ∼es ∼ed ∼ed ∼ing
subsequence (cost: 90)
 liv ␣␣␣e ␣␣␣es ␣␣␣ed ␣␣␣ed ␣␣␣ing
 ∼e ∼es ∼ed ∼ed ∼ing
submultiset (cost: 116)

ilv	␣␣e	␣␣es	␣␣ed	␣␣ed	␣␣ing, ␣i␣ng
	∼e	∼es	∼ed	∼ed	∼i∼ng, ∼ing

---

bring
bring, brings, brought, brought, bringing

substring (cost: 150)

br	␣␣ing	␣␣ings	␣␣ought	␣␣ought	␣␣inging
	∼ing	∼ings	∼ought	∼ought	∼inging

subsequence (cost: 191)

brg	␣␣in␣	␣␣in␣s	␣␣ou␣ht	␣␣ou␣ht	␣␣in␣ing, ␣␣ingin␣
	∼in∼	∼in∼s	∼ou∼ht	∼ou∼ht	∼in∼ing, ∼ingin∼

submultiset (cost: 191)

bgr	␣␣in␣	␣␣in␣s	␣␣ou␣ht	␣␣ou␣ht	␣␣in␣ing, ␣␣ingin␣
	∼in∼	∼in∼s	∼ou∼ht	∼ou∼ht	∼in∼ing, ∼ingin∼

---

write
write, writes, wrote, written, writing

substring (cost: 135)

wr	␣␣ite	␣␣ites	␣␣ote	␣␣itten	␣␣iting
	∼ite	∼ites	∼ote	∼itten	∼iting

subsequence (cost: 171)

wrt	␣␣i␣e	␣␣i␣es	␣␣o␣e	␣␣i␣ten, ␣␣it␣en	␣␣i␣ing
	∼i∼e	∼i∼es	∼o∼e	∼i∼ten, ∼it∼en	∼i∼ing

submultiset (cost: 171)

rtw	␣␣i␣e	␣␣i␣es	␣␣o␣e	␣␣i␣ten, ␣␣it␣en	␣␣i␣ing
	∼i∼e	∼i∼es	∼o∼e	∼i∼ten, ∼it∼en	∼i∼ing

---

provide
provide, provides, provided, provided, providing

substring (cost: 105)

provid	␣␣␣␣␣e	␣␣␣␣␣es	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

subsequence (cost: 147)

provid	␣␣␣␣␣e	␣␣␣␣␣es	␣␣␣␣␣ed, ␣␣␣␣de␣	␣␣␣␣␣ed, ␣␣␣␣de␣	␣␣␣␣␣ing
	∼e	∼es	∼de∼, ∼ed	∼de∼, ∼ed	∼ing

submultiset (cost: 173)

dioprv	␣␣␣␣␣e	␣␣␣␣␣es	␣␣␣␣␣ed, ␣␣␣␣de␣	␣␣␣␣␣ed, ␣␣␣␣de␣	␣␣␣␣␣ing, ␣␣␣i␣␣ng
	∼e	∼es	∼de∼, ∼ed	∼de∼, ∼ed	∼i∼ng, ∼ing

---

sit

sit, sits, sat, sat, sitting

substring (cost: 268)

s	␣it	␣its, sit␣	␣at	␣at	␣itting
	∼it	∼its, sit∼	∼at	∼at	∼itting
t	si␣	si␣s	sa␣	sa␣	si␣ting, sit␣ing
	si∼	si∼s	sa∼	sa∼	si∼ting, sit∼ing

subsequence (cost: 146)

st	␣i␣	␣i␣s	␣a␣	␣a␣	␣i␣ting, ␣it␣ing
	∼i∼	∼i∼s	∼a∼	∼a∼	∼i∼ting, ∼it∼ing

submultiset (cost: 162)

st	␣i␣	␣i␣s, si␣␣	␣a␣	␣a␣	␣i␣ting, ␣it␣ing
	∼i∼	∼i∼s, si∼	∼a∼	∼a∼	∼i∼ting, ∼it∼ing

---

stand

stand, stands, stood, stood, standing

substring (cost: 130)

st	␣␣and	␣␣ands	␣␣ood	␣␣ood	␣␣anding
	∼and	∼ands	∼ood	∼ood	∼anding

subsequence (cost: 135)

std	␣␣an␣	␣␣an␣s	␣␣oo␣	␣␣oo␣	␣␣an␣ing
	∼an∼	∼an∼s	∼oo∼	∼oo∼	∼an∼ing

submultiset (cost: 161)

dst	␣␣an␣	␣␣an␣s, s␣an␣␣	␣␣oo␣	␣␣oo␣	␣␣an␣ing
	∼an∼	∼an∼s, s∼an∼	∼oo∼	∼oo∼	∼an∼ing

lose
lose, loses, lost, lost, losing

substring (cost: 80)
 los ␣␣␣e ␣␣␣es ␣␣␣t ␣␣␣t ␣␣␣ing
 ∼e ∼es ∼t ∼t ∼ing
subsequence (cost: 101)
 los ␣␣␣e ␣␣␣es, ␣␣se␣ ␣␣␣t ␣␣␣t ␣␣␣ing
 ∼e ∼es, ∼se∼ ∼t ∼t ∼ing
submultiset (cost: 101)
 los ␣␣␣e ␣␣␣es, ␣␣se␣ ␣␣␣t ␣␣␣t ␣␣␣ing
 ∼e ∼es, ∼se∼ ∼t ∼t ∼ing

---

include
include, includes, included, included, including

substring (cost: 105)
 includ ␣␣␣␣␣␣e ␣␣␣␣␣␣es ␣␣␣␣␣␣ed ␣␣␣␣␣␣ed ␣␣␣␣␣␣ing
 ∼e ∼es ∼ed ∼ed ∼ing
subsequence (cost: 147)
 includ ␣␣␣␣␣␣e ␣␣␣␣␣␣es ␣␣␣␣␣␣ed, ␣␣␣␣␣de␣ ␣␣␣␣␣␣ed, ␣␣␣␣␣de␣ ␣␣␣␣␣␣ing
 ∼e ∼es ∼de∼, ∼ed ∼de∼, ∼ed ∼ing
submultiset (cost: 219)
 cdilnu ␣␣␣␣␣␣e ␣␣␣␣␣␣es ␣␣␣␣␣␣ed, ␣␣␣␣␣de␣ ␣␣␣␣␣␣ed, ␣␣␣␣␣de␣
 ∼e ∼es ∼de∼, ∼ed ∼de∼, ∼ed

---

pay
pay, pays, paid, paid, paying

substring (cost: 90)
 pa ␣␣y ␣␣ys ␣␣id ␣␣id ␣␣ying
 ∼y ∼ys ∼id ∼id ∼ying
subsequence (cost: 90)

pa ␣␣y ␣␣ys ␣␣id ␣␣id ␣␣ying
∼y ∼ys ∼id ∼id ∼ying
submultiset (cost: 90)
ap ␣␣y ␣␣ys ␣␣id ␣␣id ␣␣ying
∼y ∼ys ∼id ∼id ∼ying

---

meet
meet, meets, met, met, meeting

substring (cost: 191)
et me␣␣ me␣␣s m␣␣ m␣␣ me␣␣ing
me∼ me∼s m∼ m∼ me∼ing
me ␣␣et ␣␣ets ␣␣t ␣␣t ␣␣eting
∼et ∼ets ∼t ∼t ∼eting
subsequence (cost: 90)
met ␣␣e␣, ␣e␣␣ ␣␣e␣s, ␣e␣␣s ␣␣␣ ␣␣␣ ␣␣e␣ing, ␣e␣␣ing
∼e∼ ∼e∼s ∼ ∼ ∼e∼ing
submultiset (cost: 90)
emt ␣␣e␣, ␣e␣␣ ␣␣e␣s, ␣e␣␣s ␣␣␣ ␣␣␣ ␣␣e␣ing, ␣e␣␣ing
∼e∼ ∼e∼s ∼ ∼ ∼e∼ing

---

set
set, sets, set, set, setting

substring (cost: 65)
set ␣␣␣ ␣␣␣s ␣␣␣ ␣␣␣ ␣␣␣ting
∼ ∼s ∼ ∼ ∼ting
subsequence (cost: 96)
set ␣␣␣ ␣␣␣s ␣␣␣ ␣␣␣ ␣␣␣ting, ␣␣t␣ing
∼ ∼s ∼ ∼ ∼t∼ing, ∼ting
submultiset (cost: 107)
est ␣␣␣ ␣␣␣s, s␣␣␣ ␣␣␣ ␣␣␣ ␣␣␣ting, ␣␣t␣ing
∼ ∼s, s∼ ∼ ∼ ∼t∼ing, ∼ting

---

continue
continue, continues, continued, continued, continuing

substring (cost: 110)
 continu ␣␣␣␣␣␣␣e ␣␣␣␣␣␣␣es ␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣ing
 ∼e ∼es ∼ed ∼ed ∼ing
subsequence (cost: 110)
 continu ␣␣␣␣␣␣␣e ␣␣␣␣␣␣␣es ␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣ing
 ∼e ∼es ∼ed ∼ed ∼ing
submultiset (cost: 192)
 cinnotu ␣␣␣␣␣␣␣e ␣␣␣␣␣␣␣es ␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣ing, ␣␣␣␣␣n␣i␣g, ␣␣␣␣i␣␣␣ng, ␣␣␣␣in␣␣␣g, ␣␣n␣␣␣␣i␣g, ␣␣n␣i␣␣␣␣g
 ∼e ∼es ∼ed ∼ed ∼i∼ng, ∼in∼g, ∼ing, ∼n∼i∼g

---

watch
watch, watches, watched, watched, watching

substring (cost: 95)
 watch ␣␣␣␣␣ ␣␣␣␣␣es ␣␣␣␣␣ed ␣␣␣␣␣ed ␣␣␣␣␣ing
 ∼ ∼es ∼ed ∼ed ∼ing
subsequence (cost: 95)
 watch ␣␣␣␣␣ ␣␣␣␣␣es ␣␣␣␣␣ed ␣␣␣␣␣ed ␣␣␣␣␣ing
 ∼ ∼es ∼ed ∼ed ∼ing
submultiset (cost: 95)
 achtw ␣␣␣␣␣ ␣␣␣␣␣es ␣␣␣␣␣ed ␣␣␣␣␣ed ␣␣␣␣␣ing
 ∼ ∼es ∼ed ∼ed ∼ing

---

learn
learn, learns, learned, learned, learning

substring (cost: 90)
 learn ␣␣␣␣␣ ␣␣␣␣␣s ␣␣␣␣␣ed ␣␣␣␣␣ed ␣␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 116)

learn	␣␣␣␣␣	␣␣␣␣␣s	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing, ␣␣␣␣ni␣g
	∼	∼s	∼ed	∼ed	∼ing, ∼ni∼g

submultiset (cost: 158)

aelnr	␣␣␣␣␣	␣␣␣␣␣s	␣␣␣␣␣ed, ␣e␣␣␣␣d	␣␣␣␣␣ed, ␣e␣␣␣␣d	␣␣␣␣␣ing, ␣␣␣␣ni␣g
	∼	∼s	∼e∼d, ∼ed	∼e∼d, ∼ed	∼ing, ∼ni∼g

---

change

change, changes, changed, changed, changing

substring (cost: 100)

chang	␣␣␣␣␣e	␣␣␣␣␣es	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

subsequence (cost: 152)

chang	␣␣␣␣␣e	␣␣␣␣␣es	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing, ␣␣␣␣gin␣, ␣␣␣ngi␣␣
	∼e	∼es	∼ed	∼ed	∼gin∼, ∼ing, ∼ngi∼

submultiset (cost: 182)

acghn	␣␣␣␣␣e	␣␣␣␣␣es	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing, ␣␣␣␣gin␣, ␣␣␣n␣i␣g, ␣␣␣ngi␣␣
	∼e	∼es	∼ed	∼ed	∼gin∼, ∼ing, ∼n∼i∼g, ∼ngi∼

---

lead

lead, leads, led, led, leading

substring (cost: 95)

le	␣␣ad	␣␣ads	␣␣d	␣␣d	␣␣ading
	∼ad	∼ads	∼d	∼d	∼ading

subsequence (cost: 90)

led	␣␣a␣	␣␣a␣s	␣␣␣	␣␣␣	␣␣a␣ing
	∼a∼	∼a∼s	∼	∼	∼a∼ing

submultiset (cost: 90)

del	␣␣a␣	␣␣a␣s	␣␣␣	␣␣␣	␣␣a␣ing
	∼a∼	∼a∼s	∼	∼	∼a∼ing

---

stop

stop, stops, stopped, stopped, stopping

substring (cost: 100)

stop	␣␣␣␣	␣␣␣␣s	␣␣␣␣ped	␣␣␣␣ped	␣␣␣␣ping
	∼	∼s	∼ped	∼ped	∼ping

subsequence (cost: 183)

stop	␣␣␣␣	␣␣␣␣s	␣␣␣␣ped, ␣␣␣p␣ed	␣␣␣␣ped, ␣␣␣p␣ed	␣␣␣␣ping, ␣␣␣p␣ing
	∼	∼s	∼p∼ed, ∼ped	∼p∼ed, ∼ped	∼p∼ing, ∼ping

submultiset (cost: 194)

opst	␣␣␣␣	␣␣␣␣s, s␣␣␣␣	␣␣␣␣ped, ␣␣␣p␣ed	␣␣␣␣ped, ␣␣␣p␣ed	␣␣␣␣ping, ␣␣␣p␣ing
	∼	∼s, s∼	∼p∼ed, ∼ped	∼p∼ed, ∼ped	∼p∼ing, ∼ping

---

understand

understand, understands, understood, understood, understanding

substring (cost: 155)

underst	␣␣␣␣␣␣␣and	␣␣␣␣␣␣␣ands	␣␣␣␣␣␣␣ood	␣␣␣␣␣␣␣ood	␣␣␣␣␣␣␣anding
	∼and	∼ands	∼ood	∼ood	∼anding

subsequence (cost: 160)

understd	␣␣␣␣␣␣␣an␣	␣␣␣␣␣␣␣an␣s	␣␣␣␣␣␣␣oo␣	␣␣␣␣␣␣␣oo␣	␣␣␣␣␣␣␣an␣ing
	∼an∼	∼an∼s	∼oo∼	∼oo∼	∼an∼ing

submultiset (cost: 370)

ddenrstu	␣␣␣␣␣␣␣an␣, ␣n␣␣␣␣␣a␣␣	␣␣␣␣␣␣␣an␣s, ␣␣␣␣␣s␣an␣␣, ␣n␣␣␣␣␣a␣␣s, ␣n␣␣␣s␣a␣␣␣			
	∼an∼, ∼n∼a∼	∼an∼s, ∼n∼a∼s, ∼n∼s∼a∼, ∼s∼an∼			

---

follow

follow, follows, followed, followed, following

substring (cost: 95)

follow	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 95)

follow	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 95)

flloow	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

---

create

create, creates, created, created, creating

substring (cost: 100)

creat	␣␣␣␣e	␣␣␣␣es	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

subsequence (cost: 100)

creat	␣␣␣␣e	␣␣␣␣es	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

submultiset (cost: 179)

acert	␣␣␣␣e, ␣␣e␣␣	␣␣␣␣es, ␣␣e␣␣s	␣␣␣␣ed, ␣␣e␣␣d	␣␣␣␣ed, ␣␣e␣␣d	␣␣␣␣ing
	∼e, ∼e∼	∼e∼s, ∼es	∼e∼d, ∼ed	∼e∼d, ∼ed	∼ing

---

add

add, adds, added, added, adding

substring (cost: 80)

add	␣␣␣	␣␣␣s	␣␣␣ed	␣␣␣ed	␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 174)

add	␣␣␣	␣␣␣s	␣␣␣ed, ␣␣de␣, ␣d␣e␣	␣␣␣ed, ␣␣de␣, ␣d␣e␣	␣␣␣ing
	∼	∼s	∼d∼e∼, ∼de∼, ∼ed	∼d∼e∼, ∼de∼, ∼ed	∼ing

submultiset (cost: 174)

add	␣␣␣	␣␣␣s	␣␣␣ed, ␣␣de␣, ␣d␣e␣	␣␣␣ed, ␣␣de␣, ␣d␣e␣	␣␣␣ing
	∼	∼s	∼d∼e∼, ∼de∼, ∼ed	∼d∼e∼, ∼de∼, ∼ed	∼ing

---

speak

speak, speaks, spoke, spoken, speaking

substring (cost: 135)

sp	␣␣eak	␣␣eaks	␣␣oke	␣␣oken	␣␣eaking
	∼eak	∼eaks	∼oke	∼oken	∼eaking

subsequence (cost: 266)

spe	␣␣ak	␣␣aks	␣ok␣	␣ok␣n	␣␣aking
	∼ak	∼aks	∼ok∼	∼ok∼n	∼aking
spk	␣ea␣	␣ea␣s	␣o␣e	␣o␣en	␣ea␣ing
	∼ea∼	∼ea∼s	∼o∼e	∼o∼en	∼ea∼ing

submultiset (cost: 141)

ekps	␣␣a␣	␣␣a␣s, s␣␣a␣␣	␣␣o␣␣	␣␣o␣␣n	␣␣a␣ing
	∼a∼	∼a∼s, s∼a∼	∼o∼	∼o∼n	∼a∼ing

---

allow

allow, allows, allowed, allowed, allowing

substring (cost: 90)

allow	␣␣␣␣␣	␣␣␣␣␣s	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 90)

allow	␣␣␣␣␣	␣␣␣␣␣s	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 90)

allow	␣␣␣␣␣	␣␣␣␣␣s	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

---

spend

spend, spends, spent, spent, spending

substring (cost: 90)

spen	␣␣␣␣d	␣␣␣␣ds	␣␣␣␣t	␣␣␣␣t	␣␣␣␣ding
	∼d	∼ds	∼t	∼t	∼ding

subsequence (cost: 121)

spen	␣␣␣␣d	␣␣␣␣ds	␣␣␣␣t	␣␣␣␣t	␣␣␣␣ding, ␣␣␣ndi␣g
	∼d	∼ds	∼t	∼t	∼ding, ∼ndi∼g

submultiset (cost: 142)

enps	␣␣␣␣d	␣␣␣␣ds, s␣␣␣d␣	␣␣␣␣t	␣␣␣␣t	␣␣␣␣ding, ␣␣␣ndi␣g
	∼d	∼ds, s∼d∼	∼t	∼t	∼ding, ∼ndi∼g

read
read, reads, read, read, reading

substring (cost: 65)
 read ____ ____s ____ ____ ____ing
 ~ ~s ~ ~ ~ing
subsequence (cost: 65)
 read ____ ____s ____ ____ ____ing
 ~ ~s ~ ~ ~ing
submultiset (cost: 65)
 ader ____ ____s ____ ____ ____ing
 ~ ~s ~ ~ ~ing

---

walk
walk, walks, walked, walked, walking

substring (cost: 85)
 walk ____ ____s ____ed ____ed ____ing
 ~ ~s ~ed ~ed ~ing
subsequence (cost: 85)
 walk ____ ____s ____ed ____ed ____ing
 ~ ~s ~ed ~ed ~ing
submultiset (cost: 85)
 aklw ____ ____s ____ed ____ed ____ing
 ~ ~s ~ed ~ed ~ing

---

open
open, opens, opened, opened, opening

substring (cost: 85)
 open ____ ____s ____ed ____ed ____ing
 ~ ~s ~ed ~ed ~ing
subsequence (cost: 111)

open	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing, ␣␣␣ni␣g
	∼	∼s	∼ed	∼ed	∼ing, ∼ni∼g

submultiset (cost: 153)

enop	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed, ␣␣e␣␣d	␣␣␣␣ed, ␣␣e␣␣d	␣␣␣␣ing, ␣␣␣ni␣g
	∼	∼s	∼e∼d, ∼ed	∼e∼d, ∼ed	∼ing, ∼ni∼g

---

win

win, wins, won, won, winning

substring (cost: 283)

n	wi␣	wi␣s	wo␣	wo␣	wi␣ning, win␣ing, winni␣g
	wi∼	wi∼s	wo∼	wo∼	wi∼ning, win∼ing, winni∼g
w	␣in	␣ins	␣on	␣on	␣inning
	∼in	∼ins	∼on	∼on	∼inning

subsequence (cost: 182)

wn	␣i␣	␣i␣s	␣o␣	␣o␣	␣i␣ning, ␣in␣ing, ␣inni␣g
	∼i∼	∼i∼s	∼o∼	∼o∼	∼i∼ning, ∼in∼ing, ∼inni∼g

submultiset (cost: 182)

nw	␣i␣	␣i␣s	␣o␣	␣o␣	␣i␣ning, ␣in␣ing, ␣inni␣g
	∼i∼	∼i∼s	∼o∼	∼o∼	∼i∼ning, ∼in∼ing, ∼inni∼g

---

grow

grow, grows, grew, grown, growing

substring (cost: 110)

gr	␣␣ow	␣␣ows	␣␣ew	␣␣own	␣␣owing
	∼ow	∼ows	∼ew	∼own	∼owing

subsequence (cost: 115)

grw	␣␣o␣	␣␣o␣s	␣␣e␣	␣␣o␣n	␣␣o␣ing
	∼o∼	∼o∼s	∼e∼	∼o∼n	∼o∼ing

submultiset (cost: 151)

grw	␣␣o␣	␣␣o␣s	␣␣e␣	␣␣o␣n	␣␣o␣ing, g␣o␣in␣
	∼o∼	∼o∼s	∼e∼	∼o∼n	∼o∼ing, g∼o∼in∼

---

remember
remember, remembers, remembered, remembered, remembering

substring (cost: 105)
 remember ␣␣␣␣␣␣␣␣ ␣␣␣␣␣␣␣␣s ␣␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 105)
 remember ␣␣␣␣␣␣␣␣ ␣␣␣␣␣␣␣␣s ␣␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣␣ed ␣␣␣␣␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
submultiset (cost: 147)
 beeemmrr ␣␣␣␣␣␣␣␣ ␣␣␣␣␣␣␣␣s ␣␣␣␣␣␣␣␣ed, ␣␣␣␣␣␣e␣␣d, ␣␣␣e␣␣␣␣␣d, ␣e␣␣␣␣␣␣d
 ∼ ∼s ∼e∼d, ∼ed

---

offer
offer, offers, offered, offered, offering

substring (cost: 90)
 offer ␣␣␣␣␣ ␣␣␣␣␣s ␣␣␣␣␣ed ␣␣␣␣␣ed ␣␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 90)
 offer ␣␣␣␣␣ ␣␣␣␣␣s ␣␣␣␣␣ed ␣␣␣␣␣ed ␣␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
submultiset (cost: 132)
 effor ␣␣␣␣␣ ␣␣␣␣␣s ␣␣␣␣␣ed, ␣␣␣e␣␣d ␣␣␣␣␣ed, ␣␣␣e␣␣d ␣␣␣␣␣ing
 ∼ ∼s ∼e∼d, ∼ed ∼e∼d, ∼ed ∼ing

---

love
love, loves, loved, loved, loving

substring (cost: 90)
 lov ␣␣␣e ␣␣␣es ␣␣␣ed ␣␣␣ed ␣␣␣ing
 ∼e ∼es ∼ed ∼ed ∼ing
subsequence (cost: 90)

lov	␣␣␣e	␣␣␣es	␣␣␣ed	␣␣␣ed	␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

submultiset (cost: 90)

lov	␣␣␣e	␣␣␣es	␣␣␣ed	␣␣␣ed	␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

---

wait

wait, waits, waited, waited, waiting

substring (cost: 85)

wait	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

wait	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 111)

aitw	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing, ␣␣i␣␣ng
	∼	∼s	∼ed	∼ed	∼i∼ng, ∼ing

---

consider

consider, considers, considered, considered, considering

substring (cost: 105)

consider	␣␣␣␣␣␣␣␣	␣␣␣␣␣␣␣␣s	␣␣␣␣␣␣␣␣ed	␣␣␣␣␣␣␣␣ed	␣␣␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 105)

consider	␣␣␣␣␣␣␣␣	␣␣␣␣␣␣␣␣s	␣␣␣␣␣␣␣␣ed	␣␣␣␣␣␣␣␣ed	␣␣␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 312)

cdeinors	␣␣␣␣␣␣␣	␣␣␣␣␣␣␣s, ␣␣s␣␣␣␣	␣␣␣␣␣␣␣ed, ␣␣␣␣␣e␣␣d, ␣␣␣␣␣d␣␣e␣, ␣␣␣␣␣de␣␣␣	
	∼	∼s, ∼s∼	∼d∼e∼, ∼de∼, ∼e∼d, ∼ed	

---

buy

buy, buys, bought, bought, buying

substring (cost: 261)

b	␣uy	␣uys	␣ought	␣ought	␣uying
	∼uy	∼uys	∼ought	∼ought	∼uying
u	b␣y	b␣ys	bo␣ght	bo␣ght	b␣ying
	b∼y	b∼ys	bo∼ght	bo∼ght	b∼ying

subsequence (cost: 120)

bu	␣␣y	␣␣ys	␣o␣ght	␣o␣ght	␣␣ying
	∼y	∼ys	∼o∼ght	∼o∼ght	∼ying

submultiset (cost: 120)

bu	␣␣y	␣␣ys	␣o␣ght	␣o␣ght	␣␣ying
	∼y	∼ys	∼o∼ght	∼o∼ght	∼ying

---

appear

appear, appears, appeared, appeared, appearing

substring (cost: 95)

appear	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 95)

appear	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 137)

aaeppr	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed, ␣␣␣e␣␣␣d	␣␣␣␣␣␣ed, ␣␣␣e␣␣␣d	␣␣␣␣␣␣ing
	∼	∼s	∼e∼d, ∼ed	∼e∼d, ∼ed	∼ing

---

serve

serve, serves, served, served, serving

substring (cost: 95)

serv	␣␣␣␣e	␣␣␣␣es	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

subsequence (cost: 95)

serv	␣␣␣␣e	␣␣␣␣es	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

submultiset (cost: 210)

ersv	␣␣␣e, ␣e␣␣	␣␣␣es, ␣e␣␣s, s␣␣e␣, se␣␣␣	␣␣␣ed, ␣e␣␣d	␣␣␣ed, ␣e␣␣d	␣␣␣ing
	∼e, ∼e∼	∼e∼s, ∼es, s∼e∼, se∼	∼e∼d, ∼ed	∼e∼d, ∼ed	∼ing

---

die

die, dies, died, died, dying

substring (cost: 253)

d	␣ie	␣ies	␣ied, die␣	␣ied, die␣	␣ying
	∼ie	∼ies	∼ied, die∼	∼ied, die∼	∼ying
i	d␣e	d␣es	d␣ed	d␣ed	dy␣ng
	d∼e	d∼es	d∼ed	d∼ed	dy∼ng

subsequence (cost: 90)

di	␣␣e	␣␣es	␣␣ed	␣␣ed	␣y␣ng
	∼e	∼es	∼ed	∼ed	∼y∼ng

submultiset (cost: 132)

di	␣␣e	␣␣es	␣␣ed, d␣e␣	␣␣ed, d␣e␣	␣y␣ng
	∼e	∼es	∼ed, d∼e∼	∼ed, d∼e∼	∼y∼ng

---

stay

stay, stays, stayed, stayed, staying

substring (cost: 85)

stay	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

stay	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 96)

asty	␣␣␣␣	␣␣␣␣s, s␣␣␣␣	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s, s∼	∼ed	∼ed	∼ing

---

fall

fall, falls, fell, fallen, falling

substring (cost: 115)
 ll fa␣␣ fa␣␣s fe␣␣ fa␣␣en fa␣␣ing
 fa∼ fa∼s fe∼ fa∼en fa∼ing
subsequence (cost: 120)
 fll ␣a␣␣ ␣a␣␣s ␣e␣␣ ␣a␣␣en ␣a␣␣ing
 ∼a∼ ∼a∼s ∼e∼ ∼a∼en ∼a∼ing
submultiset (cost: 120)
 fll ␣a␣␣ ␣a␣␣s ␣e␣␣ ␣a␣␣en ␣a␣␣ing
 ∼a∼ ∼a∼s ∼e∼ ∼a∼en ∼a∼ing

---

build
build, builds, built, built, building

substring (cost: 90)
 buil ␣␣␣␣d ␣␣␣␣ds ␣␣␣␣t ␣␣␣␣t ␣␣␣␣ding
 ∼d ∼ds ∼t ∼t ∼ding
subsequence (cost: 90)
 buil ␣␣␣␣d ␣␣␣␣ds ␣␣␣␣t ␣␣␣␣t ␣␣␣␣ding
 ∼d ∼ds ∼t ∼t ∼ding
submultiset (cost: 126)
 bilu ␣␣␣␣d ␣␣␣␣ds ␣␣␣␣t ␣␣␣␣t ␣␣␣␣ding, ␣␣i␣d␣ng
 ∼d ∼ds ∼t ∼t ∼ding, ∼i∼d∼ng

---

send
send, sends, sent, sent, sending

substring (cost: 85)
 sen ␣␣␣d ␣␣␣ds ␣␣␣t ␣␣␣t ␣␣␣ding
 ∼d ∼ds ∼t ∼t ∼ding
subsequence (cost: 116)
 sen ␣␣␣d ␣␣␣ds ␣␣␣t ␣␣␣t ␣␣␣ding, ␣␣ndi␣g
 ∼d ∼ds ∼t ∼t ∼ding, ∼ndi∼g
submultiset (cost: 137)

ens	␣␣␣d	␣␣␣ds, s␣␣d␣	␣␣␣t	␣␣␣t	␣␣␣ding, ␣␣ndi␣g
	∼d	∼ds, s∼d∼	∼t	∼t	∼ding, ∼ndi∼g

---

cut
cut, cuts, cut, cut, cutting

substring (cost: 65)

cut	␣␣␣	␣␣␣s	␣␣␣	␣␣␣	␣␣␣ting
	∼	∼s	∼	∼	∼ting

subsequence (cost: 96)

cut	␣␣␣	␣␣␣s	␣␣␣	␣␣␣	␣␣␣ting, ␣␣t␣ing
	∼	∼s	∼	∼	∼t∼ing, ∼ting

submultiset (cost: 96)

ctu	␣␣␣	␣␣␣s	␣␣␣	␣␣␣	␣␣␣ting, ␣␣t␣ing
	∼	∼s	∼	∼	∼t∼ing, ∼ting

---

expect
expect, expects, expect, expected, expecting

substring (cost: 85)

expect	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼	∼ed	∼ing

subsequence (cost: 85)

expect	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼	∼ed	∼ing

submultiset (cost: 122)

ceeptx	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣	␣␣␣␣␣␣ed, ␣␣␣e␣␣␣d, e␣␣␣␣␣␣d	␣␣␣␣␣␣ing
	∼	∼s	∼	∼e∼d, ∼ed, e∼d	∼ing

---

kill
kill, kills, killed, killed, killing

substring (cost: 85)

kill	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 85)

kill	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 111)

ikll	␣␣␣␣	␣␣␣␣s	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing, ␣i␣␣␣ng
	∼	∼s	∼ed	∼ed	∼i∼ng, ∼ing

---

suggest

suggest, suggests, suggested, suggested, suggesting

substring (cost: 100)

suggest	␣␣␣␣␣␣␣	␣␣␣␣␣␣␣s	␣␣␣␣␣␣␣ed	␣␣␣␣␣␣␣ed	␣␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 100)

suggest	␣␣␣␣␣␣␣	␣␣␣␣␣␣␣s	␣␣␣␣␣␣␣ed	␣␣␣␣␣␣␣ed	␣␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 200)

eggsstu	␣␣␣␣␣␣␣	␣␣␣␣␣␣␣s, ␣␣␣␣␣s␣␣, s␣␣␣␣␣␣␣	␣␣␣␣␣␣␣ed, ␣␣␣␣e␣␣␣d	␣␣␣␣␣␣␣ed, ␣␣␣␣e␣␣␣d
	∼	∼s, ∼s∼, s∼	∼e∼d, ∼ed	∼e∼d, ∼ed

---

reach

reach, reaches, reached, reached, reaching

substring (cost: 95)

reach	␣␣␣␣␣	␣␣␣␣␣es	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼	∼es	∼ed	∼ed	∼ing

subsequence (cost: 95)

reach	␣␣␣␣␣	␣␣␣␣␣es	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼	∼es	∼ed	∼ed	∼ing

submultiset (cost: 158)

acehr	␣␣␣␣␣	␣␣␣␣␣es, ␣e␣␣␣␣s	␣␣␣␣␣ed, ␣e␣␣␣␣d	␣␣␣␣␣ed, ␣e␣␣␣␣d	␣␣␣␣␣ing
	∼	∼e∼s, ∼es	∼e∼d, ∼ed	∼e∼d, ∼ed	∼ing

---

remain

remain, remains, remained, remained, remaining

substring (cost: 95)

remain	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 147)

remain	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing, ␣␣␣␣␣ni␣g, ␣␣␣␣in␣␣g
	∼	∼s	∼ed	∼ed	∼in∼g, ∼ing, ∼ni∼g

submultiset (cost: 214)

aeimnr	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed, ␣e␣␣␣␣d	␣␣␣␣␣␣ed, ␣e␣␣␣␣d	␣␣␣␣␣␣ing, ␣␣␣␣␣ni␣g, ␣␣␣␣i␣␣ng, ␣␣␣␣in␣␣g
	∼	∼s	∼e∼d, ∼ed	∼e∼d, ∼ed	∼i∼ng, ∼in∼g, ∼ing, ∼ni∼g

---

require

require, requires, required, required, requiring

substring (cost: 105)

requir	␣␣␣␣␣␣e	␣␣␣␣␣␣es	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

subsequence (cost: 105)

requir	␣␣␣␣␣␣e	␣␣␣␣␣␣es	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼e	∼es	∼ed	∼ed	∼ing

submultiset (cost: 210)

eiqrru	␣␣␣␣␣␣e, ␣e␣␣␣␣	␣␣␣␣␣␣es, ␣e␣␣␣␣s	␣␣␣␣␣␣ed, ␣e␣␣␣␣d	␣␣␣␣␣␣ed, ␣e␣␣␣␣d	␣␣␣␣␣␣ing, ␣␣␣␣i␣␣ng
	∼e, ∼e∼	∼e∼s, ∼es	∼e∼d, ∼ed	∼e∼d, ∼ed	∼i∼ng, ∼ing

---

thank

thank, thanks, thanked, thanked, thanking

substring (cost: 90)

thank	␣␣␣␣␣	␣␣␣␣␣s	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 90)

thank	␣␣␣␣␣	␣␣␣␣␣s	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 121)

ahknt	␣␣␣␣␣	␣␣␣␣␣s	␣␣␣␣␣ed	␣␣␣␣␣ed	␣␣␣␣␣ing, ␣␣␣n␣i␣g
	∼	∼s	∼ed	∼ed	∼ing, ∼n∼i∼g

---

report

report, reports, reported, reported, reporting

substring (cost: 95)

report	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

subsequence (cost: 95)

report	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed	␣␣␣␣␣␣ed	␣␣␣␣␣␣ing
	∼	∼s	∼ed	∼ed	∼ing

submultiset (cost: 137)

eoprrt	␣␣␣␣␣␣	␣␣␣␣␣␣s	␣␣␣␣␣␣ed, ␣e␣␣␣␣␣d	␣␣␣␣␣␣ed, ␣e␣␣␣␣␣d		␣␣␣␣␣␣ing
	∼	∼s	∼e∼d, ∼ed	∼e∼d, ∼ed		∼ing

---

sell

sell, sells, sold, sold, selling

substring (cost: 360)

l	se␣l, sel␣	se␣ls, sel␣s	so␣d	so␣d	se␣ling, sel␣ing
	se∼l, sel∼	se∼ls, sel∼s	so∼d	so∼d	se∼ling, sel∼ing
s	␣ell	␣ells, sell␣	␣old	␣old	␣elling
	∼ell	∼ells, sell∼	∼old	∼old	∼elling

subsequence (cost: 213)

sl	␣e␣l, ␣el␣	␣e␣ls, ␣el␣s	␣o␣d	␣o␣d	␣e␣ling, ␣el␣ing
	∼e∼l, ∼el∼	∼e∼ls, ∼el∼s	∼o∼d	∼o∼d	∼e∼ling, ∼el∼ing

submultiset (cost: 259)

ls	␣e␣l, ␣el␣	␣e␣ls, ␣el␣s, se␣l␣, sel␣␣	␣o␣d	␣o␣d	␣e␣ling, ␣el␣ing
	∼e∼l, ∼el∼	∼e∼ls, ∼el∼s, se∼l∼, sel∼	∼o∼d	∼o∼d	∼e∼ling, ∼el∼ing

---

pull

pull, pulls, pulled, pulled, pulling

substring (cost: 85)
 pull ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
subsequence (cost: 85)
 pull ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing
submultiset (cost: 85)
 llpu ␣␣␣␣ ␣␣␣␣s ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼s ∼ed ∼ed ∼ing

---

raise
raise, raises, raised, raised, raising

substring (cost: 95)
 rais ␣␣␣␣e ␣␣␣␣es ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼e ∼es ∼ed ∼ed ∼ing
subsequence (cost: 116)
 rais ␣␣␣␣e ␣␣␣␣es, ␣␣␣se␣ ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼e ∼es, ∼se∼ ∼ed ∼ed ∼ing
submultiset (cost: 142)
 airs ␣␣␣␣e ␣␣␣␣es, ␣␣␣se␣ ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing, ␣␣i␣␣ng
 ∼e ∼es, ∼se∼ ∼ed ∼ed ∼i∼ng, ∼ing

---

pass
pass, passes, passed, passed, passing

substring (cost: 90)
 pass ␣␣␣␣ ␣␣␣␣es ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼es ∼ed ∼ed ∼ing
subsequence (cost: 137)
 pass ␣␣␣␣ ␣␣␣␣es, ␣␣␣se␣, ␣␣s␣e␣ ␣␣␣␣ed ␣␣␣␣ed ␣␣␣␣ing
 ∼ ∼es, ∼s∼e∼, ∼se∼ ∼ed ∼ed ∼ing
submultiset (cost: 137)

apss	␣␣␣␣	␣␣␣␣es, ␣␣␣se␣, ␣␣s␣e␣	␣␣␣␣ed	␣␣␣␣ed	␣␣␣␣ing
	∼	∼es, ∼s∼e∼, ∼se∼	∼ed	∼ed	∼ing

REFERENCES

Adam Albright and Bruce Hayes. Modeling English past tense intuitions with minimal generalization. In M. Maxwell, editor, *Proceedings of the 6th meeting of the ACL Special Interest Group in Computational Phonology*. ACL, Philadelphia, 2002.

Adam Albright and Bruce Hayes. Rules versus analogy in English past tenses: a computational/experimental study. *Cognition*, 90:119–161, 2003.

Afra Alishahi. *Computational Model of Human Language Acquisition*. Morgan & Claypool Publishers, 2010.

Stephen R. Anderson. Where's morphology? *Linguistic Inquiry*, 13(4):571–612, 1982.

Stephen R. Anderson. *A-Morphous Morphology*. Cambridge University Press, Cambridge, 1992.

Mark Aronoff. *Morphology by itself: stems and inflectional classes*. MIT Press, Cambridge, MA, 1994.

Mark Aronoff and Kirsten Fudeman. *What is Morphology?* Wiley, 2nd edition, 2011.

R. Harald Baayen. *Word Frequency Description*, volume 18 of *Text, Speech, and Language Technology*. Kluwer, Dordrecht, 2001.

Matthew Baerman. Paradigmatic chaos in Nuer. *Language*, 88(3):467–494, 2012.

Mark C. Baker. *Lexical categories: Verbs, nouns and adjectives*, volume 102. Cambridge University Press, Cambridge, 2003.

Michele Banko and Robert C Moore. Part of speech tagging in context. In *Proceedings of the 20th international conference on Computational Linguistics*, page 556. Association for Computational Linguistics, 2004.

Marco Baroni, Johannes Matiasek, and Harald Trost. Unsupervised discovery of morphologically related words based on orthographic and semantic similarity. In *Proceedings of the ACL-02 workshop on Morphological and phonological learning-Volume 6*, pages 48–57. Association for Computational Linguistics, 2002.

Laurie Bauer. *Morphological productivity*, volume 95. Cambridge University Press, Cambridge, 2001.

Laurie Bauer. *Introducing Linguistic Morphology*. Georgetown University Press, Washington, D.C., 2nd edition, 2003.

Laurie Bauer. *A Glossary of Morphology*. Edinburgh University Press, Edinburgh, 2004.

Steven Bird, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

James P. Blevins. Word-based morphology. *Journal of Linguistics*, 42(3):531–573, 2006.

James P. Blevins. *Word and Paradigm Morphology*. Oxford University Press, Oxford, 2013.

Bernard Bloch. English verb inflection. *Language*, 23(4):399–418, 1947.

Harry Bochner. *Simplicity in Generative Morphology*. Mouton de Gruyter, Berlin, 1992.

Geert Booij. *Construction Morphology*. Oxford University Press, Oxford, 2010.

Claudia Borg and Albert Gatt. Crowd-sourcing evaluation of automatically acquired, morphologically related word groupings. In *Proceedings of the International Conference on Language Resources and Evaluation 2014*, 2014.

Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2301–2309, 2011.

Dunstan Brown and Andrew Hippisley. *Network Morphology: A Defaults-based Theory of Word Structure*. Cambridge University Press, Cambridge, 2012.

Roger Brown. *A First Language: The Early Stages*. Harvard University Press, 1973.

Joan L. Bybee. *Morphology: A Study of the Relation between Meaning and Form*. John Benjamins, Amsterdam/Philadelphia, 1985.

Andrew Carstairs. Paradigm economy. *Journal of Linguistics*, 19:115–125, 1983.

Andrew Carstairs. *Allomorphy in Inflexion*. Croom Helm, London, 1987.

Erwin Chan. Learning probabilistic paradigms for morphology in a latent class model. In *Proceedings of the Eighth Meeting of the ACL Special Interest Group on Computational Phonology at HLT-NAACL 2006*, pages 69–78. New York City, 2006.

Noam Chomsky. *Syntactic Structures*. Mouton de Gruyter, Berlin, second edition, 1957/2002.

Noam Chomsky. *Aspects of the Theory of Syntax*. MIT press, 1965.

Noam Chomsky. Remarks on nominalization. In Roderick A. Jacobs and Peters S. Rosenbaum, editors, *Readings in English Transformational Grammar*, pages 184–221. Ginn and Company, Waltham, Mass., 1970.

Noam Chomsky. *Rules and representations*, volume 3. Cambridge University Press, 1980.

Noam Chomsky and Morris Halle. *The Sound Pattern of English*. Harper and Row, New York, 1968.

Christos Christodoulopoulos, Sharon Goldwater, and Mark Steedman. Two decades of unsupervised POS induction: How far have we come? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2010.

Jon Claerbout and Martin Karrenbach. Electronic documents give reproducible research a new meaning. In *Proc. 62nd Ann. Int. Meeting of the Soc. of Exploration Geophysics*, pages 601–604, 1992.

Alexander Clark. Inducing syntactic categories by context distribution clustering. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 91–94. Association for Computational Linguistics, 2000.

Alexander Clark. Combining distributional and morphological information for part of speech induction. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 1*, pages 59–66. Association for Computational Linguistics, 2003.

Alexander Clark. Distributional learning of syntax. In Nick Chater, Alexander Clark, John A. Goldsmith, and Amy Perfors, editors, *Empiricism and Language Learnability*. Oxford University Press, Oxford, 2015.

Alexander Clark and Shalom Lappin. Unsupervised learning and grammar induction. In Alexander Clark, Chris Fox, and Shalom Lappin, editors, *Handbook of Computational Linguistics and Natural Language Processing*, pages 197–220. Wiley-Blackwell, Oxford, 2010.

Greville G. Corbett and Norman M. Fraser. Network Morphology: a DATR account of Russian nominal inflection. *Journal of Linguistics*, 29:113–142, 1993.

Angelo Roth Costanzo. *Romance Conjugational Classes: Learning from the Peripheries*. PhD thesis, The Ohio State University, 2011.

Mathias Creutz. Unsupervised segmentation of words using prior distributions of morph length and frequency. In *Proceedings of the ACL 2003*, pages 280–287. 2003.

Mathias Creutz and Krista Lagus. Inducing the morphological lexicon of a natural language from unannotated text. In *Proceedings of AKRR'05, International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning*, pages 106–113. 2005.

David Crystal. *A Dictionary of Linguistics and Phonetics*. Wiley-Blackwell, 6th edition, 2008.

Antje Dammel. How – and why – do inflectional classes arise? A case study on Swedish and Norwegian conjugation. In Fabio Montermini, Gilles Boyé, and Jesse Tseng, editors, *Selected Proceedings of the 6th Décembrettes*, pages 12–21. Cascadilla Proceedings Project, Somerville, MA, 2009.

Ferdinand de Saussure. *Cours de linguistique générale*. 1916.

Rose-Marie Anne Déchaine. *Predicates across categories: towards a category-neutral syntax*. PhD thesis, University of Massachusetts, Amherst, 1993.

Laura J. Downing, T. Alan Hall, and Renate Raffelsiefen, editors. *Paradigms in Phonological Theory*. Oxford University Press, Oxford, 2004.

Markus Dreyer. *A non-parametric model for the discovery of inflectional paradigms from plain text using graphical models over strings*. PhD thesis, Johns Hopkins University, 2011.

Markus Dreyer and Jason Eisner. Discovering morphological paradigms from plain text using a Dirichlet process mixture model. In *Proceedings of Empirical Methods in Natural Language Processing*, pages 616–627. 2011.

Raffaella Folli and Christiane Ulbrich, editors. *Interface in Linguistic: New Research Perspectives*. Oxford University Press, Oxford, 2010.

Paul L. Garvin. *On Linguistic Method*. Mouton & Co., The Hague, 1964.

John A. Goldsmith. *Autosegmental Phonology*. PhD dissertation, MIT, 1976.

John A. Goldsmith. *Autosegmental and metrical phonology*. Basil Blackwell, Oxford, 1990.

John A. Goldsmith. Linguistica: An automatic morphological analyzer. In John Boyle, Jung-Hyuck Lee, and Arika Okrent, editors, *Papers from the 36th Annual Meeting of the Chicago Linguistic Society, Main Session*. Chicago Linguistic Society, Chicago, 2000.

John A. Goldsmith. Unsupervised learning of the morphology of a natural language. *Computational Linguistics*, 27(2):153–198, 2001.

John A. Goldsmith. From algorithms to generative grammar and back again. In *Proceedings of the 40th Annual Meeting of the Chicago Linguistic Society*. Chicago Linguistic Society, Chicago, 2004.

John A. Goldsmith. An algorithm for the unsupervised learning of morphology. *Natural Language Engineering*, 12(4):353–371, 2006.

John A. Goldsmith. Morphological analogy: Only a beginning. In James P. Blevins and Juliette Blevins, editors, *Analogy in Grammar: Form and Acquisition*, pages 138–164. Oxford University Press, Oxford, 2009.

John A. Goldsmith. Segmentation and morphology. In Alexander Clark, Chris Fox, and Shalom Lappin, editors, *Handbook of Computational Linguistics and Natural Language Processing*, pages 364–393. Wiley-Blackwell, Oxford, 2010.

John A. Goldsmith. The evaluation metric in generative grammar, December 2011a. Paper presented at the 50th anniversary celebration for the MIT Department of Linguistics.

John A. Goldsmith. A group structure for strings: Towards a learning algorithm for morphophonology. Technical Report TR-2011-06, Department of Computer Science, University of Chicago, 2011b.

John A. Goldsmith and Jason Riggle. Information theoretic approaches to phonological structure: the case of Finnish vowel harmony. *Natural Language & Linguistic Theory*, 30 (3):859–896, 2012.

John A. Goldsmith and Xiuli Wang. Word manifolds. Manuscript, University of Chicago, 2012.

John A. Goldsmith and Aris Xanthos. Learning phonological categories. *Language*, 85(1): 4–38, 2009.

John A. Goldsmith, Jackson L. Lee, and Aris Xanthos. Computational learning of morphology. *Annual Review of Linguistics*, 3:85–106, 2017.

Sharon Goldwater and Tom Griffiths. A fully Bayesian approach to unsupervised part-of-speech tagging. In *Annual Meeting of the Association for Computational Linguistics*, volume 45, page 744, 2007.

Aria Haghighi and Dan Klein. Prototype-driven learning for sequence models. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 320–327. Association for Computational Linguistics, 2006.

Morris Halle. Phonology in Generative Grammar. *Word*, 18:54–72, 1962.

Morris Halle and Alec Marantz. Distributed Morphology and the pieces of inflection. In Kenneth Hale and Samuel Jay Keyser, editors, *The View from Building 20: Essays in Linguistics in Honor of Sylvain Bromberger*, pages 111–176. MIT Press, Cambridge, MA, 1993.

Harald Hammarström and Lars Borin. Unsupervised learning of morphology. *Computational Linguistics*, 37(2):309–350, 2011.

Zellig S. Harris. From phoneme to morpheme. *Language*, 31(2):190–222, 1955.

Martin Haspelmath. *Understanding Morphology*. Arnold, London, 1st edition, 2002.

Martin Haspelmath and Andrea D. Sims. *Understanding Morphology*. Hodder Education, London, 2nd edition, 2010.

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. Tier-based Strictly Local Constraints for Phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: Short Papers - Volume 2*, HLT '11, pages 58–64, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-88-6. URL `http://dl.acm.org/citation.cfm?id=2002736.2002750`.

Charles F Hockett. Componential analysis of Sierra Popoluca. *International Journal of American Linguistics*, 13(4):258–267, 1947.

Charles F. Hockett. Two models of grammatical description. *Word*, 10:210–34, 1954.

Charles F Hockett. The origin of speech. *Scientific American*, 203:88–111, 1960.

Mans Hulden, M. Forsberg, and M. Ahlberg. Semi-supervised learning of morphological paradigms and lexicons. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 569–578. Gothenburg, Sweden, 2014.

John D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 2007. doi: 10.1109/MCSE.2007.55.

Sharon Inkelas. *The Interplay of Morphology and Phonology*. Oxford University Press, Oxford, 2014.

Ray Jackendoff. *X-bar syntax*. The MIT Press, 1977.

John T. Jensen. *Morphology: Word Structure in Generative Grammar*. John Benjamins, Amsterdam/Philadelphia, 1990.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An introduction to natural language processing, computational linguistics, and speech recognition*. 2006.

Aleksandr E. Kibrik. Archi. In Andrew Spencer and Arnold M. Zwicky, editors, *The Handbook of Morphology*, pages 455–76. Blackwell, Oxford, 1998.

Henry Kučera and W. Nelson Francis. *Computational Analysis of Present-Day American English*. Brown University Press, Providence, 1967.

D. Robert Ladd. *Simultaneous Structure in Phonology*. Oxford University Press, Oxford, 2014.

Hun-Tak Thomas Lee and Colleen Wong. Cancorp: the Hong Kong Cantonese Child Language Corpus. *Cahiers de Linguistique Asie Orientale*, 27:211–228, 1998.

Jackson L. Lee. Automatic morphological alignment and clustering. Technical Report TR-2014-07, Department of Computer Science, University of Chicago, May 2014.

Jackson L. Lee. Morphological Paradigms: Computational Structure and Unsupervised Learning. In *Proceedings of NAACL-HLT 2015 Student Research Workshop (SRW)*, pages 161–167, Denver, Colorado, June 2015a. Association for Computational Linguistics.

Jackson L. Lee. PyCantonese: Cantonese linguistic research in the age of big data. Talk at the Childhood Bilingualism Research Centre, the Chinese University of Hong Kong, 2015b.

Jackson L. Lee and John A. Goldsmith. Linguistica 5: Unsupervised Learning of Linguistic Structure. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics*, San Diego, California, June 2016a. Association for Computational Linguistics.

Jackson L. Lee and John A. Goldsmith. Complexity across morphological paradigms: a minimum description length approach to identifying inflectional stems. In *Proceedings of the MorphologyFest*, 2016b.

Jackson L. Lee, Ross Burkholder, Gallagher B. Flinn, and Emily R. Coppess. Working with CHAT transcripts in Python. Technical Report TR-2016-02, Department of Computer Science, University of Chicago, 2016.

Man-Tak Leung, Sam-Po Law, and Suk-Yee Fung. Type and token frequencies of phonological units in Hong Kong Cantonese. *Behavior Research Methods, Instruments, and Computer*, 36(3):500–505, 2004.

Rochelle Lieber. *Introducing Morphology*. Cambridge University Press, Cambridge, 2010.

Brian MacWhinney. *The CHILDES project: Tools for analyzing talk*. Lawrence Erlbaum Associates, Mahwah, NJ, 2000.

Peter H. Matthews. *Inflectional Morphology*. Cambridge University Press, Cambridge, 1972.

Peter H. Matthews. *Morphology*. Cambridge University Press, Cambridge, 2nd edition, 1991.

Peter H. Matthews. *Concise Dictionary of Linguistics*. Oxford University Press, 2nd edition, 2007.

Mike Maxwell. Electronic grammars and reproducible research. In Sebastian Nordhoff, editor, *Language Documentation & Conservation Special Publication No. 4*, pages 207–234. University of Hawai'i Press, 2012.

John J. McCarthy. *Formal problems in Semitic phonology and morphology*. PhD dissertation, MIT, 1979.

John J. McCarthy and Alan Prince. Faithfulness and reduplicative identity. In Jill Beckman, Suzanne Urbanczyk, and Laura Walsh Dickey, editors, *University of Massachusetts Occasional Papers in Linguistics 18: Papers in Optimality Theory*, pages 249–384. 1995.

Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

Bernard Merialdo. Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–171, 1994.

Ian M. Mitchell, Randall J. LeVeque, and Victoria Stodden. Reproducible research for scientific computing: Tools and strategies for changing the culture. *Computing in Science and Engineering*, 14(4):13–17, 2012. ISSN 1521-9615. doi: http://doi.ieeecomputersociety.org/10.1109/MCSE.2012.38.

Gereon Müller. Notes on paradigm economy. *Morphology*, 17:1–38, 2007.

Tomonori Nagano and Virginia Valian. Is fully-automated corpus-based language acquisition research feasible? Poster presentation at the Architectures and Mechanisms of Language Processing (AMLaP), 2011.

Eugene A. Nida. *Morphology: the descriptive analysis of words*. University of Michigan publications, Ann Arbor, 1949.

Partha Niyogi. *The Computational Nature of Language Learning and Evolution*. MIT press, Cambridge, MA; London, 2006.

Ted Pedersen. Empiricism is not a matter of faith. *Computational Linguistics*, 34(3):465–470, 2008.

Fernando Pérez and Brian E. Granger. IPython: A System for Interactive Scientific Computing. *Computing in Science & Engineering*, 9(3):21–29, 2007. doi: 10.1109/MCSE.2007.53.

Mike Pham and Jackson L. Lee. Combining successor and predecessor frequencies to model truncation in Brazilian Portuguese. Technical Report TR-2014-15, Department of Computer Science, University of Chicago, October 2014.

Mike Pham and Jackson L. Lee. Mincing words: Balancing recovery and deletion in word truncation. *Glossa: A Journal of General Linguistics*, 3(1):36, 2018.

Florian Prinz, Thomas Schlange, and Khusru Asadullah. Believe it or not: how much can we rely on published data on potential drug targets? *Nature Reviews Drug Discovery*, 10 (9):712, 2011.

Geoffrey K. Pullum and Barbara C. Scholz. Empirical assessment of stimulus poverty arguments. *The Linguistic Review*, 18(1-2):9–50, 2002.

Gillian Ramchand and Charles Reiss, editors. *The Oxford Handbook of Linguistic interfaces*. Oxford University Press, Oxford, 2007.

Jason Riggle. The role of similarity in non-local dependencies. Presented at the Workshop on Information-theoretic Approaches to Linguistics, LSA Linguistic Institute 2011, University of Colorado, Boulder., 2011.

Jorma Rissanen. *Stochastic Complexity in Statistical Inquiry*, volume 15 of *Series in computer science*. World Scientific, Singapore; Teaneck, N.J., 1989.

Edward Sapir. *Language: An introduction to the study of speech*. Harcourt, Brace and Company, New York, 1921.

Patrick Schone and Daniel Jurafsky. Knowledge-free induction of morphology using latent semantic analysis. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 67–72. Association for Computational Linguistics, 2000.

Patrick Schone and Daniel Jurafsky. Knowledge-free induction of inflectional morphologies. In *Proceedings of the second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies*, pages 1–9. Association for Computational Linguistics, 2001.

Hinrich Schütze. Distributional part-of-speech tagging. In *Proceedings of the seventh conference on European chapter of the Association for Computational Linguistics*, pages 141–148. Morgan Kaufmann Publishers Inc., 1995.

Noah A. Smith and Jason Eisner. Contrastive estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 354–362. Association for Computational Linguistics, 2005.

Andrew Spencer. *Morphological Theory*. Basil Blackwell, Oxford, England, 1991.

Andrew Spencer. Identifying stems. *Word Structure*, 5(1):88–108, 2012.

Andrew Spencer. *Lexical Relatedness: A Paradigm-based Model*. Oxford University Press, Oxford, 2013.

Donca Steriade. Correspondence and the phonological lexicon. Lectures at the LSA Summer Linguistic Institute 2009, University of California, Berkeley., 2009.

Gregory T. Stump. *Inflectional Morphology: A Theory of Paradigm Structure*. Cambridge University Press, Cambridge, 2001a.

Gregory T. Stump. Default inheritance hierarchies and the evolution of inflectional classes. In Laurel Brinton, editor, *Historical Linguistics 1999*. Benjamins, Amsterdam, 2001b.

Gregory T. Stump and Raphael A. Finkel. *Morphological Typology: From Word to Paradigm*. Cambridge University Press, Cambridge, 2013.

R. L. Trask. *Key Concepts in Language and Linguistics*. Routledge, London, 1999.

Guido van Rossum and Fred L Drake Jr. *Python reference manual*. Centrum voor Wiskunde en Informatica, Amsterdam, 1995.

Aline Villavicencio, Thierry Poibeau, Anna Korhonen, and Afra Alishahi, editors. *Cognitive Aspects of Computational Language Acquisition*. Springer, 2013.

Qin Iris Wang and Dale Schuurmans. Improved estimation for unsupervised part-of-speech tagging. In *Natural Language Processing and Knowledge Engineering, 2005. IEEE NLP-KE'05. Proceedings of 2005 IEEE International Conference on*, pages 219–224. IEEE, 2005.

Michael Waskom. Seaborn: Statistical data visualization in Python, v0.6.0, 2015.

Jelte M. Wicherts and Marjan Bakker. Publish (your data) or (let the data) perish! Why not publish your data too? *Intelligence*, 40(2):73–76, 2012.

David Yarowsky and Richard Wicentowski. Minimally supervised morphological analysis by multimodal alignment. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 207–216. Association for Computational Linguistics, 2000.

Virginia Yip and Stephen Matthews. *The Bilingual Child: Early Development and Language Contact*. Cambridge University Press, 2007.

Alan C. L. Yu. *A Natural History of Infixation*. Oxford University Press, Oxford, 2007.

Daniel Zeman. Unsupervised acquiring of morphological paradigms from tokenized text. In *Advances in Multilingual and Multimodal Information Retrieval: 8th Workshop of the Cross-Language Evaluation Forum, CLEF 2007*, pages 892–899. Budapest, 2008.