Cross-Formalism Minimum Description Length

Marina Ermolaeva John A. Goldsmith

October 2023

1 Introduction

1.1 Background

In this paper we imagine a way to use some ideas about algorithmic complexity to develop a method for comparing linguistic theories. We do more than just imagine it, though; we take some steps towards making it a reality, and we propose in the end to encourage linguists to engage with us in comparing linguistic theories in a quantitative fashion.

The over-all idea is a development that one of us proposed in skeletal fashion a number of years ago (Goldsmith 2015), which was itself grounded in work on minimum description length (MDL) work, by Rissanen (1989) in general, and by de Marcken (1996) in particular. In some respects our perspectives have been heavily influenced by Chomsky's early work (1975 [1955], 1965) in which he suggested that a method for evaluating grammars (an *evaluation metric*) was an essential part of a linguistic theory.[1]

We begin with one methodological premise: we require that methods of writing grammars be rich enough that they can describe any set of observed linguistic data.[2] The consequence of this is in a sense simple: the linguist (or whoever is writing the grammar) is not free to ignore or reject some

[1] Though he dropped this perspective as early as (Chomsky and Lasnik 1977)—he moved towards a "principles and parameters" style of analysis, which was conceptually simpler, but too simple, in our opinion—he may have taken steps in this direction in more recent work; this is not clear.

[2] This is a soft sort of requirement: it does not mean that we should be required to account for data that is literally made up. It means that we should be required to account for data that is produced by a linguist in good faith. Since that notion is a bit hazy, the requirement is equally hazy—but in practice, its application should be reasonably clear. Here is an example that illustrates the force of the principle. It is widely assumed in analyses of the English auxiliary that there is a class of modal auxiliary verbs that includes *may, must, can, could,* and *might* and that no more than one of these verbs can appear in a given finite clause. There are dialects of American English that permit the sequence *might could*, however. A theory of grammar must be rich enough to allow for the description of these facts, even if the facts appear only in an unfamiliar dialect.

data, though she may construct a grammar in which some of the data is treated as falling outside the otherwise regular patterns in the language as described by the grammar. Following Goldsmith (2015), we view this methodological requirement as a commitment to an *empiricist* methodology: one may dislike some data, but that does not allow one to discard it. If some linguistic data which one's grammar failed to deal with elegantly eventually proves to be in error, then that grammar will (as we will see) profit from the elimination of inaccurate data.

In general, linguistic theories that are rich enough to deal with observed data are also rich enough to allow for many grammars generating the same data, and it is this simple fact that motivates the inclusion of something like a Chomskian evaluation metric as an addendum to a theory's architecture describing how grammars are to be formulated.

In Chomsky's early conception, the evaluation metric would be available to be used to rank all grammars that correctly generated the observed sentences of a language. In one formulation, the evaluation metric might be as simple as counting the number of symbols, and in such a case, the best grammar (within a theory) would be the one with the smallest number of symbols consistent with the theory and with the data.

There were two concerns with this picture from what we think of as "classical" generative linguistics: first, it seemed to require very considerable work on constructing (so to speak) a grammar of grammars, which would be a set of overarching generalizations that lay on top of grammars of actual languages; and second, it seemed to allow for a sort of pathological backdoor. If all the best grammar needed to do was generate the correct utterances of a language, then a very simple grammar will always do the trick. It was therefore also necessary to ensure that the grammar failed to generate ungrammatical sentences.

Yet in real-life virtually never did two grammatical descriptions arise of the same set of data. The closest thing that arose was that some participants in controversies would claim that two analyses (typically in distinct and competing frameworks) were notational variants, but no effort was made to prove this, nor to see whether an evaluation metric could adjudicate in selecting one of the competing proposals.

We believe that linguistic theory could be improved by the inclusion of *minimum description length* (MDL) computations, and that this would allow not only to argue for analyses within specific

frameworks, it can be used to compare different frameworks. That is what we aim to illustrate in this paper, and we will explain what MDL is in the next section.

This paper illustrates what a competition of the sort we are interested in would look like. We do not actually put *two different theories* head to head here, for practical reasons: we recognize that this would be a major project, and would require at least two teams of workers. What we do instead is illustrate how certain ways of analyzing strings can be compared quantitatively, and take this somewhat artificial example as one which can show how different linguistic theories defended by different people could be subjected to the same kinds of comparisons.

1.2 What is MDL?

Minimum Description Length (MDL) analysis is a statistical framework that employs some of the basic ideas of information theory to calculate both the complexity of a given account of some data (which means, for a linguist, the complexity of a grammar), and the degree to which the account models the data well.

In addition, MDL offers a way of performing both calculations in a way that ends up measuring them in the same units, the *bit*. Since we want to minimize the complexity of the account, and maximize the degree to which an account fits the data, MDL proposes that we add the two measurements together, and seek the analysis which optimizes the sum. In short, the best analysis is the one that is shortest and fits the data the best, in full recognition that the two desiderata typically tug in different directions. Let us consider each of these two terms: complexity and goodness of fit to data.

The analyst always chooses the simplest analysis—as Ockham's razor has long insisted—but the challenge is to find a way to compare the simplicity (or its inverse, complexity) of a given analysis. The field of algorithmic complexity (established by Solomonoff (1964), Kolmogorov (1965), and Chaitin (1966)) offers a major step in meeting the challenge by identifying the simplest analysis as the shortest program that describes or embodies the analysis, given a specific programming language that is rich enough to include any program at all.

Such a statement immediately gives rise to two questions: in what sense does an analysis have a "shortest" version? Why should an analysis have shorter or longer versions? And doesn't the

language in which we express these programs have a big influence on which version is shorter and which is longer?

Whether one is a linguist or a computer programmer, one understands that an analysis can be made smaller — tighter, some might say — by looking to find subparts that are used more than once, and abstracting away from them (by writing separate functions, for example, to take care of a computation that is done more than once). Conversely, it's always *possible* to make an analysis longer, by adding something to it that ultimately has no impact on the output at all. One might then well wonder how one ever knows that a version of an analysis is truly the shortest implementation one will ever be able to find? All we really know at any given time is the shortest version we have come up with; only very rarely can we know that there truly is no shorter version possible.

From our point of view — which is scientific, rather than mathematical — such concerns are natural but they are not roadblocks. At any given moment, we must be satisfied with the shortest implementation *that we have found*, recognizing that in the future there is always the possibility that a shorter implementation will be found. Since we will implement the competition between frameworks as a competition between human teams, each team will have a vested interest in finding the shortest implementation of their own system.

And doesn't the choice of a linguistic theory have a major impact on how short or long an implementation of an analysis will turn out to be?

Here MDL offers a surprising answer. MDL says this: yes, choosing a particular linguistic theory may make some computations much shorter. But the length of linguistic theory that allows for this must be added to the length of the entire implementation that we are looking at. MDL says that we must choose as a bedrock foundation a programming language that can compute anything at all (a "universal Turing machine"—a "UTM"), and then length of the compiler for the linguistic theory that is being used will simply be added in to compute the total length of the analysis. From a linguist's point of view, the system that compiles from the language of grammars to a universal Turing machine is a "linguistic theory", which tells us how grammars may be written, and then explains how a statement in a grammar is implemented in the most basic of terms. And MDL says that that linguistic theory is also a program, and its length must be added to the length of the analysis.

But MDL also allows for the possibility that the linguist cares about the most, the case where certain elements of computation occur in language after language. In these extremely common cases, what we have to bear in mind is that the linguistic theory needs to be stated only once, and its effects will be found in simplifications that are available in many or all of the various languages that are analyzed. The effects of this simplicity are in effect "mortgaged" over a range of grammars, each of which uses such a case.

Thus the bottom line is that in order for a linguist to use MDL for the purposes we have in mind, it is necessary to treat not just a single language, but rather grammars from *multiple* languages—the more, the better. The length of the analyses for each language will consist, then, of the sum of the linguistic theory plus the sum of all of the analyses/grammars needed to account for the data.

Pulling these strands together, we see that the bedrock reality for MDL analysis is a choice of a UTM (universal Turing machine) or its equivalent, a (universal) programming language (all familiar programming languages are universal in the relevant sense). The linguist may develop a UG (universal grammar) which is written in the universal programming language, allowing the grammar to contain abbreviations used often ("often" either within a language or across languages). The linguist creates a grammar (perhaps with the assistance of a computer, or the assistance of human beings) written within their UG (or if they choose a trivial UG, directly in the programming language of the UTM).

What makes MDL possible is the insight that grammar length and goodness of fit of grammar to data are both expressible in terms of bits, and MDL says in particular that we would like to minimize this sum. There are two words that are often used in this context: we speak of the *cost* of a grammar or of some data, and we sometimes talk of the *length* of a grammar (and less often of the length of a corpus, which is the data). We use the term *cost* to express the number of bits needed for a grammar, or assigned to a given corpus, and we use the term length, when speaking of a grammar, of the number of characters. The difference between cost and length is normally only a specific factor, which is the number of bits that is used to express each character, which is a number typically a little more than 6 (since 2^6 is 64, and our alphabet is usually a bit bigger than 64).

1.3 Linguistic theories

What is a linguistic theory? The point of view we take in this paper is highly formal, but we will try to show how the elements that we emphasize connect up with practical reality for the linguist. A linguistic theory consists of a set of assumptions that we hold fixed, or take for granted, before we begin our work on a new language (or, just as likely, before we begin to work on an old language where we pretend that no work has been done before). It is everything we hold to be true about languages in general, or perhaps better, what we hold to be required in the tools we have available for describing the grammar of the language.

A linguistic theory is, we might say, the grammar of grammars, and it lives, therefore, at a rather high level of abstraction. But we would like to say that it has two principal components. One of these is familiar to linguists; the other is less familiar to linguists (though familiar to computational linguists), and it is necessary because of the nature of our project, which is based on providing an account of a *given* corpus. The corpus is given by the organizers of the event, so to speak—just as data is provided by nature to the scientist. Therefore the participants in the event must have an explicit method of showing how the data is handled within their theory. In a sense, this is nothing new; a defender of a given theory is always in principle responsible for knowing whether their theory and their grammar does or doesn't generate a given sentence. What we add to that is the requirement that this responsibility must be made explicit (which is what we mean by requiring an algorithm for its implementation). We will sometimes use the term "parser" to designate this algorithm or program.

Returning to the components, then, of a linguistic theory, there are two—the parser which we have just discussed, and the more familiar part, which is a statement of what can and what must be in any given grammar of a natural language. When linguists think about creating or modifying a linguistic theory, they are thinking about ways of enriching or paring down the kinds of objects that can be used in formulating a grammar. It is important for us to underscore, though, that there are some subtleties here—for it is always possible that the step that one linguists takes, of adding a concept to their linguistic theory, is viewed by others as illegitimate because the concept should (in their view) be part of the language-particular grammar. Two simple examples: should linguistic theory include grammatical relations for arguments of a verb? Some think yes; some think no. A

decision ultimately needs to be made. Another example of disagreement concerns the number of lexical categories. Some think that there is a preset number, which is simply to say, that it is a matter settled by the linguistic theory. Others—certainly including computational linguists, in the great majority—allow for individual grammars to specify lexical categories such as proper nouns, or even more specifically family names and given names, or days of the week and months of the year.

Putting the same point a slightly different way, linguistic theory can and does serve as a place where a linguist can put things about the languages that they study so that they do not need to justify the inclusion of certain things in the individual grammars they write. This practice will not disappear, but it is only helpful to emphasize that when one linguist does this for reasons that appear to them eminently reasonable, another linguist may look at the move and say that it is sneaky and unjustified.

On the other hand, it is not unreasonable to consider that a linguistic theory ought to contain all linguistic universals that we know of. That does leave them unexplained, but it not unreasonably allows for grammars to be shorter. Still, this question can lead to considerably controversy, and we will not delve into it here.

From the MDL point of view, to analyze a word or sentence within a linguistic theory is to encode it using a grammar written within the assumptions of the particular linguistic theory. This results in a compressed form of the word or sentence, and as the name suggests, it really should be shorter than the raw data.

Why should that be so? Because each utterance in a language respects the structure of that language, and this structure can also be viewed (as if through the eyes of information theory) as *redundancy,* and redundancy can be *removed*, leaving behind a shorter string of symbols.

As we have said now, a UG should contain instructions on how to analyze a member of a corpus given a compatible grammar. That is, the UG must contain what we have called a "parser," which takes as input a grammar for a given corpus C and a word $w \in C$ and outputs the encoding of w given the grammar.

From an MDL point of view, which focuses on the question of which overall analysis is *better*, it is actually the *length* of the encoding that matters, since that is what we are interested in when calculating the "description length", rather than the binary string itself which is the encoding (Grünwald 2007). This may seem unnatural at first, but it is so.

In practice, the linguists appeal to UG as a means of *explaining* certain aspects of particular languages, and this amounts to saying that there are truths about a language that are based not on specifics that lie in the formulation of that language's grammar, but on something deeper than the grammar—and that something is the Universal Grammar.

So the linguist places some specifications into the UG, and thereby frees the language-particular grammar from the responsibility of making those specifications. In this way, UG can and will be a way of simplifying—making shorter—the language's grammar.

All of this is good, and inevitable for the linguist developing a formal theory of language. But there is a downside as well. It is possible to shift more and more from the language-particular grammars into the UG without being aware of it, and by doing so to present an incorrect impression of how simple one's grammar is for particular languages.

Let us step back for a moment and put together the three pieces we have talked about: first, the compression of the linguistic data that is the responsibility of the grammar; second, the axiom of MDL, that we must minimize the *sum* of two things, the compressed length of the data and the grammar; and third, linguists want to factor what they believe about languages into two parts, one of which is particular to the language (what we call its *grammar*) and that which is universal, which we call UG, and we take it to be universal: available to all languages. The question now is to ask whether from MDL's point of view the length of the grammar is length of the language-particular grammar, or does it also include the UG?

Our answer is that the length of the grammar, from an MDL perspective, is the sum of the lengths of the language-particular grammar and the length of the UG (in a phrase: *there's no such thing as a free UG*), but we do not have to "pay" for it each time we analyze a language. Our goal is to globally minimize the sum of the compressed length of the data and the length of the grammar, but if we do this for a set of several or many languages, the length of the UG is incurred only once.

It is worth noting that this approach says nothing about learning, or how the language-specific grammars are obtained. This is no different from classical MDL: while it can be used as a metric in grammar induction (to name a few examples, Goldsmith 2001, 2006, Hu et al. 2005, Rasin and Katzir 2016, Rasin et al. 2018), the grammars themselves do not record the history of how they were obtained; rather, learning occurs through comparing multiple grammars and procceding with the best candidate(s). One advantage of this approach is the ability to compare hand-written

grammars. This is especially important when considering ideas from mainstream linguistics, where it is quite uncommon to devise a learning procedure producing the analysis that is being proposed. The downside, of course, is that learnability is not taken into account.

1.4 Experiment setup

Summarizing the above, the version of MDL we are interested in is the sum of the following, given a UTM and a set of corpora:

1. UG term: the cost of the universal grammar on the chosen UTM, expressed in bits;
2. Grammar term: the sum of costs of individual grammars that have been developed for all the chosen corpora, in bits. Linguists may prefer to think of these as grammars of the chosen languages, but we require only that the grammars be sufficient for the corpora in particular;
3. Data term: the sum of compressed lengths of all corpora in bits.

In what follows, we will refer to this measure as *cross-formalism MDL* or *three-term MDL*, and to the standard version as *two-term MDL*.

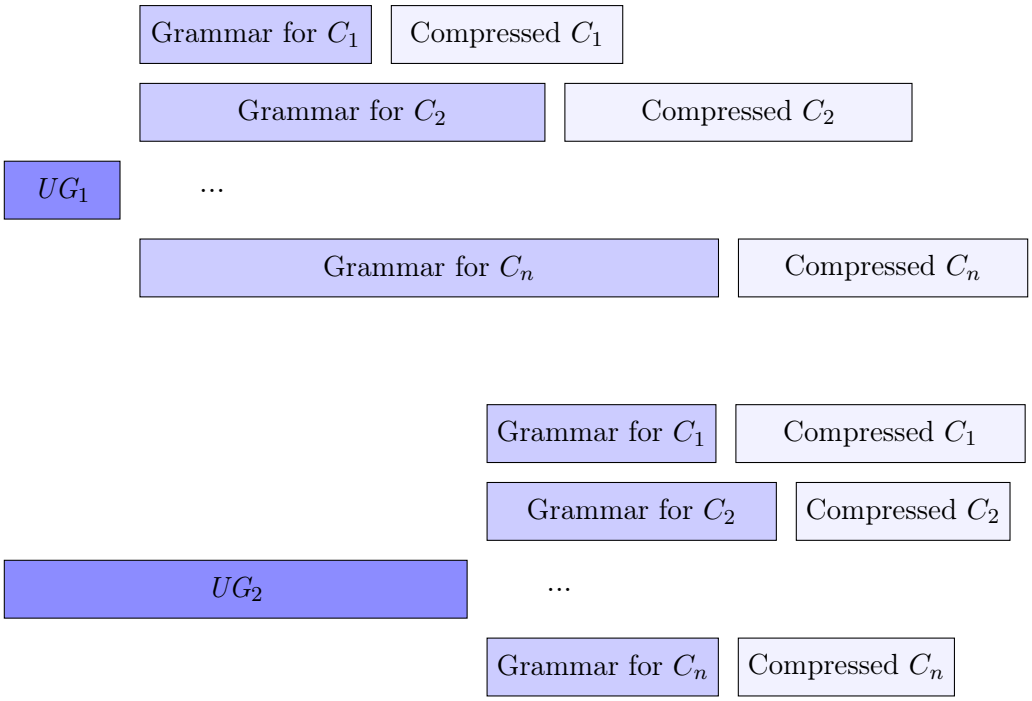

Figure 1: Cross-formalism MDL

The addition of the UG term is what allows us to compare not only different grammars, but distinct linguistic theories. A schematic illustration is given in Figure 1. Let us consider two linguistic theories, UG_1 and UG_2, and a set of n corpora $C_1, C_2, ..., C_n$ drawn from different natural languages. For each UG, the total cost is the sum of $2n+1$ terms representing the UG, n grammars, and the compressed representations of all n corpora, each encoded with the appropriate grammar.

As proof of concept, we perform a specific instance of this task. As originally envisioned by Goldsmith (2015), the comparison is implemented as a competition between formal linguistic theories. Given a fixed UTM[3] and a set of corpora, the participants must provide a single UG and grammars for each corpus. The entry with the lowest total cost wins, and winning, in this context, means that the UG of the winner has won.

In what follows, we set up a competition for a number of formalisms, including nine formally robust but linguistically naive baselines and the formalism for encoding morphological structure behind Linguistica (Goldsmith 2001, 2006; Lee and Goldsmith 2016). We make an explicit decision to focus on information contained within words — or, from the linguistic perspective, on phonological and morphological patterns.

Our encoding conventions are as follows. For now, we adopt a very basic fixed-length encoding scheme for grammars. Assume a "universal" alphabet A_U, which is the union of the 100 ASCII printable and whitespace characters and the set of all characters occuring in the corpora. The length (in bits) of each UG and corpus-specific grammar, taken as a string of characters, is calculated as $\log_2(|A_U|)$ multiplied by the length of the grammar in characters.

Each UG calculates the length of compressed data for each corpus using the respective grammar.[4] To reflect our decision to focus on phonology and morphology, instead of requiring the grammars to encode the corpus as a single sequence of characters (with or without separators between words),

[3]The original idea involved a fixed set of approved UTMs, with a procedure allowing the participants to propose new ones. We are adopting a special case of this, where there is exactly one approved UTM. The choice of the UTM can (at least in theory — but see Subsection 3.1) impact the results of the competition. For example, a linguistic theory based on finite-state automata would benefit from having an implementation of automata already included in the UTM, whereas one relying on tree structures would have a shorter encoding if it did not have to implement (and encode) these structures from scratch. In the interest of keeping the competition both fair and transparent, the chosen UTM is Python 3.6. Furthermore, all entries are limited to the standard library with the addition of some mathematical functions.

[4]A point of concern in this respect is ensuring that the compressed length is calculated correctly and consistently by each UG. One way to ensure this is to require the UGs to produce the actual binary string encoding the corpus (rather than just the cost) and demonstrate that the original data can be recovered from it. For now, the corpus cost is made transparent via detailed and easily verifiable word-by-word calculations for a small corpus (see Section 2.)

we frame the task as a sequence of tasks of encoding a single word. That is, the corpus is considered a multiset of words, and its cost is calculated as the sum of costs for each individual word. Under these conditions, grammars can access and compress patterns relying on information within a word, but not that between words in a sentence or larger chunk of text.[5]

A Python 3 implementation of three-term MDL, along with UGs (parsers) for all baselines and Linguistica and sample grammars, can be found at https://github.com/mermolaeva/cross-formalism-mdl.

2 Grammar formalisms

This section introduces the baseline formalisms and Linguistica. The baselines fall into two broad categories regarding their grammar terms. "Small" grammars grow in the size of the alphabet. They encode patterns involving characters within a window of a fixed size and therefore have a tendency to overgenerate. From a linguistic point of view, they can be considered models of phonology. On the other hand, "large" grammars capture relations between characters within the entire word, and therefore grow in the number of word types in the corpus. These grammars can be thought of as models of morphology.

Under our straightforward encoding scheme for the UG and individual grammars, the UG and grammar terms are easy to calculate. However, the corpus cost calculation is specific to each formalism and much less obvious. For transparency, in this section we demonstrate how the formalisms compress the data on the small corpus given in Figure 2. This toy corpus contains 15 tokens and 12 types and uses an alphabet of 14 symbols (including #).

dog#	*jabberwocky#*	*laugh#*	*lion#*
dog#	*jump#*	*laugh#*	*lions#*
dog#	*jumping#*	*laughing#*	*zzzzz#*
dogs#	*jumps#*	*laughs#*	

Figure 2: A toy corpus

[5] In particular, this approach would rob non-linguistic compression algorithms like (Ziv and Lempel 1977), which rely on identifying repeating substrings of data, of most of their advantage. This appears reasonable in the context of finding patterns *within* individual words.

2.1 Notation

An *alphabet* Σ is defined as a finite set of characters. The size of an alphabet, or the number of characters it contains, is written as $|\Sigma|$. A *string* or *word* over an alphabet Σ is a finite sequence composed of elements from the set Σ. Similar to alphabet size, the length (in characters) of a string w is represented as $|w|$. For any two strings u and v, their *concatenation* is written as uv. Given a string $w = uv$, u is a *prefix* of w and v is a *suffix* of w. Σ^* is the set of all strings over the alphabet Σ, including the *empty string* ϵ of length 0.

Let C denote a *corpus*, which is a collection of words. Assume that each word explicitly ends with the separator character #. A_C is the alphabet containing all unique characters occurring in C, including #. T_C denotes the list of *types*, or unique words in C. The number of times a string s occurs throughout the corpus is denoted as $\mathbf{Count}_C(s)$. For simplicity, we will suppress the subscript and use A, T, and **Count** rather than A_C, T_C, and $\mathbf{Count}_C$ where it does not lead to ambiguity. Furthermore, we use $\hat{}$ to denote the start of a word. For example, $\mathbf{Count}(\hat{}a)$ is the number of times a occurs word-initially in the corpus. In particular, for any $w \in C$, $\mathbf{Count}(\hat{}w)$ is the *multiplicity*, or number of instances, of the whole word w in the corpus.

2.2 Small grammars

2.2.1 Unigrams

A basic way to encode strings, which we will refer to as a *promiscuous* grammar, can be described as "anything goes". Such a grammar assumes that all characters that occur in the corpus are equally likely to appear in any position in the string, making all strings of a given length equally easy to describe. A promiscuous grammar itself is simply a list of characters in A (Figure 3). There is no need to include the separator # explicitly, since we can assume that it exists in every grammar; however, it is still necessary to include it in the UG term by adding instructions on how to process this character correctly.

a, b, c, d, e, g, h, i, j, k, l, m, n, o, p, r, s, u, w, y, z

Figure 3: A promiscuous grammar

This approach results in an extremely low grammar cost. However, since there are no restrictions on what characters can appear in what positions in a word, each character has to be selected from $|A|$ equiprobable options, so encoding a word w costs $|w| \times \log_2 |A|$ bits. The cost of encoding each word depends only on its length. For instance, for the word *dog#* it is $4 \times \log_2(22) \approx 17.838$ bits; and this cost has to be paid separately for each instance of the word in the corpus — in this case, three times.

Word	**Cost**	**Word**	**Cost**	**Word**	**Cost**	**Word**	**Cost**
dog#	17.838×3	*jump#*	22.297	*laugh#*	26.757×2	*lion#*	22.297
dogs#	22.297	*jumping#*	35.675	*laughing#*	40.135	*lions#*	26.757
jabberwocky#	53.513	*jumps#*	26.757	*laughs#*	31.216	*zzzzz#*	26.757
						Total:	414.727

Table 1: Toy corpus cost using the promiscuous grammar (Figure 3)

A refinement of this approach is the *unigram* grammar, which still treats the characters as independent from each other but takes into account how frequently each of them appears in the corpus, assigning lower encoding costs to those that occur more often. While the promiscuous grammar implicitly assumes that the occurrence of characters follows a uniform distribution, the unigram grammar assigns explicit probabilities to characters based on their observed *frequencies* in the corpus. The frequency of a character can be obtained from such a grammar by dividing its count by the sum of counts of all characters in the corpus. To make this information recoverable from the grammar, we record how many times each character occurred in the corpus (Figure 4).

(15), *a* (5), *b* (2), *c* (1), *d* (4), *e* (1), *g* (10), *h* (4), *i* (4), *j* (4), *k* (1),
l (6), *m* (3), *n* (4), *o* (7), *p* (3), *r* (1), *s* (4), *u* (7), *w* (1), *y* (1), *z* (5)

Figure 4: A unigram grammar

This change produces a reduction of the corpus cost. While it is too small to offset the increased grammar cost for the toy corpus, the effect is much more pronounced on larger amounts of data. The cost of encoding a character of frequency f in bits is given by $-\log_2 f$. Thus, the total cost associated with a corpus word w is as follows:

$$\sum_{i=1}^{|w|} -\log_2 \left(\frac{\textbf{Count}(w_i)}{\sum_{x \in A} \textbf{Count}(x)} \right) \tag{1}$$

Word	Cost	Word	Cost	Word	Cost	Word	Cost
dog#	14.12 × 3	*jump#*	20.812	*laugh#*	22.292 × 2	*lion#*	19.397
dogs#	18.66	*jumping#*	33.107	*laughing#*	34.587	*lions#*	23.936
jabberwocky#	65.434	*jumps#*	25.351	*laughs#*	26.831	*zzzzz#*	23.718
						Total:	378.777

Table 2: Toy corpus cost using unigrams (Figure 4)

2.2.2 Bigrams

This character-based approach can be further refined by moving from unigrams to *bigrams*, or substrings of length 2. The bigram grammar itself is a list of bigrams (and word-initial unigrams) annotated with counts (Figure 5).

ˆ*d* (4); ˆ*j* (4); ˆ*l* (6); ˆ*z* (1); *ab* (1), *au* (4); *bb* (1), *be* (1); *ck* (1); *do* (4); *er* (1); *g#* (5), *gh* (4), *gs* (1); *h#* (2), *hi* (1), *hs* (1); *in* (2), *io* (2); *ja* (1), *ju* (3); *ky* (1); *la* (4), *li* (2); *mp* (3); *n#* (1), *ng* (2), *ns* (1); *oc* (1), *og* (4), *on* (2); *p#* (1), *pi* (1), *ps* (1); *rw* (1); *s#* (4); *ug* (4), *um* (3); *wo* (1); *y#* (1); *z#* (1), *zz* (4)

Figure 5: A bigram grammar

The bigram grammar is quadratic, rather than linear, in the size of A, but yields an even better compression of the corpus. To encode a word w in a bigram grammar, we calculate the probability of each character w_i as the count of the bigram $w_{i-1}w_i$ divided by the sum of counts of all bigrams starting with w_i:

$$-\log_2 \left(\frac{\textbf{Count}(\hat{}w_1)}{\sum_{x \in A} \textbf{Count}(\hat{}x)} \right) + \sum_{i=2}^{|w|} -\log_2 \left(\frac{\textbf{Count}(w_{i-1}w_i)}{\sum_{x \in A} \textbf{Count}(w_{i-1}x)} \right) \tag{2}$$

Word	Cost	Word	Cost	Word	Cost	Word	Cost
dog#	3.714×3	*jump#*	5.129	*laugh#*	5.358×2	*lion#*	7.714
dogs#	6.036	*jumping#*	8.129	*laughing#*	9.358	*lions#*	7.714
jabberwocky#	11.036	*jumps#*	5.129	*laughs#*	6.358	*zzzzz#*	7.517
						Total:	95.98

Table 3: Toy corpus cost using bigrams (Figure 5)

A generalization of this approach is the n-gram model, where the probability of the next character is conditioned on $n-1$ previous characters. The n-gram model can be thought of as a probabilistic version of strictly local grammars, which are sets of substrings of length n that are permitted (or, in an equivalent formulation, forbidden) in strings drawn from the corpus. From a linguistic perspective, n-grams can capture phenomena where the number of intervening characters between the trigger and the target is no greater than n. Bigrams in particular are good enough for local processes where the trigger and the target are adjacent, such as word-final devoicing.

2.3 Large grammars

2.3.1 Lists and trees

The simplest way to record a finite number of strings is to store them as a *list* (Figure 6). Under this approach, the length of the grammar (in characters) amounts to the sum of lengths of all types in the corpus.

dog#, *dogs#*, *jabberwocky#*, *jump#*, *jumps#*, *jumping#*,
laugh#, *laughs#*, *laughing#*, *lion#*, *lions#*, *zzzzz#*

Figure 6: A list of types

Encoding each word amounts to selecting one of T options, yielding the flat cost of $\log_2(T)$ bits per word in the corpus. For our toy corpus of 15 tokens and 12 types, each word takes $\log_2 12 \approx 3.585$ bits, and the corpus total is $15 \times \log_2 12 \approx 53.774$ bits.

Moving away from treating words as unanalyzable units, we recognize that the strings in a natural language corpus have a significant overlap. We may attempt to capitalize on common substrings by utilizing a more efficient data structure such as a *trie* (Fredkin 1960). A prefix trie

(Figure 7) is a tree structure for storing a set of strings drawn from some alphabet. It contains one internal node for every prefix of any of the strings, with the root node associated with the empty string (here labeled with our start-of-word character, ˆ) and other internal nodes storing individual characters from the alphabet. Leaf nodes store the character #. All leaves dominated by a given node share a common prefix associated with that node. An alternative, which we will not explore here, is a suffix trie, where nodes correspond to common suffixes instead of prefixes.

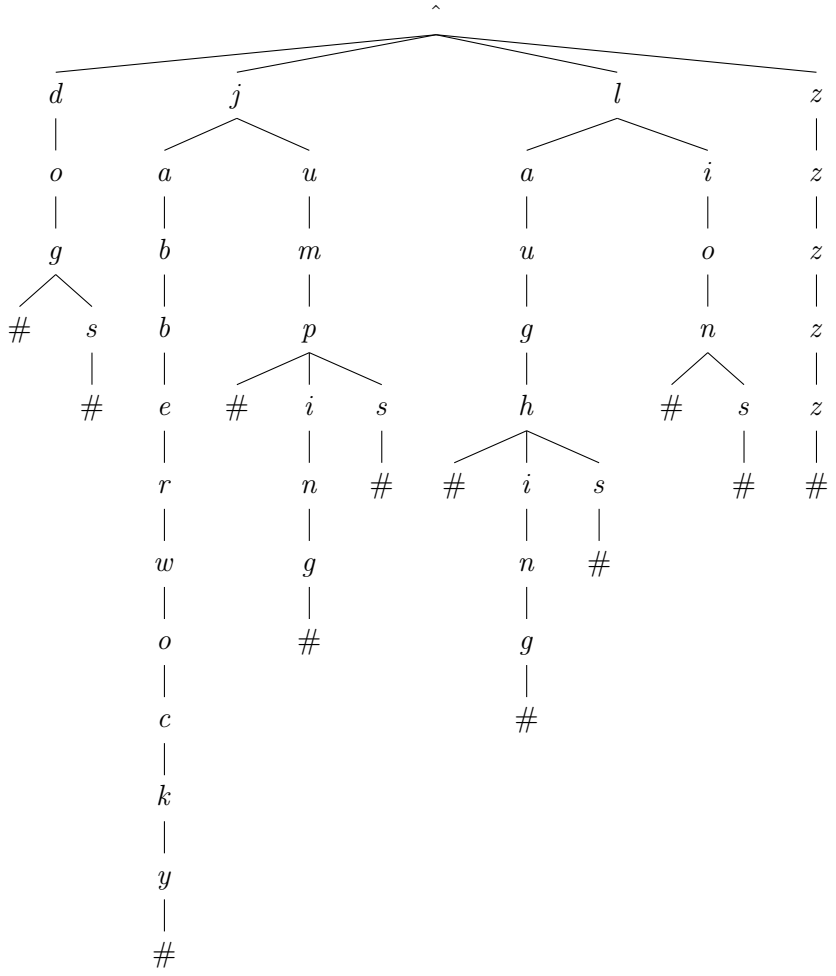

Figure 7: A trie grammar

On the grammar encoding side, a trie avoids some of the redundancy of the list. For a largely suffixal language such as English, a trie will compress the roots, since many of them correspond to

common prefixes and will be encoded only once in the trie. The tradeoff is having to keep track of the tree structure, which involves additional characters at each nesting level.[6]

To encode a corpus word w, we have to record the path in the trie from the root to the leaf corresponding to w. We traverse the trie starting from the root. For each character w_i in the word, the choices are limited to the children of the node corresponding to the prefix $w_1...w_{i-1}$ — or, in other words, the number of distinct options for the character following the prefix $w_1...w_{i-1}$ that exist in the corpus. This can be formalized as shown in (3).

$$\sum_{i=1}^{|w|} |\{x | x \in A \ \& \ \exists \alpha : \alpha \in A^* \ \& \ w_1...w_{i-1}x\alpha \in T\}| \tag{3}$$

Note that following a tree edge that is a single child costs $\log_2(1) = 0$ bits. For example, encoding *dog#* involves selecting the initial *d* out of four options, paying $log_2(4) = 2$ bits, and selecting the final # over *s* for $log_2(2) = 1$ bit, for a total of 3 bits.

Word	**Cost**	**Word**	**Cost**	**Word**	**Cost**	**Word**	**Cost**
dog#	3.0×3	*jump#*	4.585	*laugh#*	4.585×2	*lion#*	4.0
dogs#	3.0	*jumping#*	4.585	*laughing#*	4.585	*lions#*	4.0
jabberwocky#	3.0	*jumps#*	4.585	*laughs#*	4.585	*zzzzz#*	2.0
						Total:	57.095

Table 4: Toy corpus cost using the trie (Figure 7)

An optimization of this structure is a *radix tree* (Morrison 1968), in which the nodes can contain strings longer than one character, and each non-leaf node has at least two children. A radix tree can be obtained from a trie by merging each node that is a single child with its parent (Figure 8).

[6] Consider, for example, a compressed (and not particularly human-readable) representation of the grammar in Figure 7. This is a traversal of the trie in preorder, with the children of each node enclosed in curly braces:

```
{d{o{g{#s{#}}}}l{i{o{n{#s{#}}}}a{u{g{h{#s{#}i{n{g{#}}}}}}}}j{u{m{p{#s{#}i{n{g{#}}}}}}}}
```

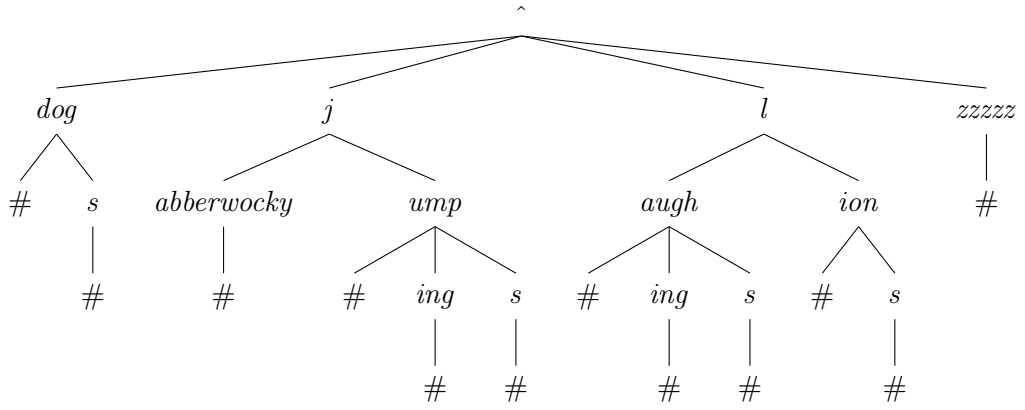

Figure 8: A radix tree grammar

A radix tree has less structure to encode, and therefore a lower grammar cost. The corpus cost is exactly the same as when using a trie, because single children don't contribute to the cost.

2.3.2 Adding counts

As with the promiscuous grammar vs. unigrams, a further refinement takes into account the frequencies of strings in the corpus. For the list formalism, we can record how many times each word occurred in the corpus and store this information in the grammar, as in Figure 9.

dog# (3), *dogs#* (1), *jabberwocky#* (1), *jump#* (1), *jumps#* (1), *jumping#* (1),

laugh# (2), *laughs#* (1), *laughing#* (1), *lion#* (1), *lions#* (1), *zzzzz#* (1)

Figure 9: A list with counts

Once again, adding counts to the grammar increases the grammar cost. The cost of encoding a word w is simply $-\log_2$ of its frequency in the corpus:

$$-\log_2\left(\frac{\mathbf{Count}(\hat{}w)}{\sum_{t\in T}\mathbf{Count}(\hat{}t)}\right) \tag{4}$$

In the toy corpus, the word *dog*, which occurs three times, is the cheapest to encode at $-\log_2\frac{3}{15}\approx 2.322$ bits. The word *laugh* occurs twice and requires $-\log_2\frac{2}{15}\approx 2.907$ bits per instance, and the rest of the words cost $-\log_2\frac{1}{15}\approx 3.907$ bits each. The entire corpus encoding is approximately 51.848 bits.

Tries and radix trees can also be annotated with counts, as shown in Figure 10 and Figure 11 respectively. It is easy to see that once we add counts, the corpus cost for both tries and radix trees is exactly the same as when using a list. Consider an arbitrary string $a\beta\# \in C$, where $a \in A$ and $\beta \in A^*$. In a list with counts, in order to encode an instance of this string, we obtain its frequency by dividing the number of times it occurs in the corpus by the total number of strings in the corpus:

$$-\log_2 \left(\frac{\textbf{Count}(\hat{}a\beta\#)}{|C|} \right) \tag{5}$$

Now consider a trie with counts, where the cost is distributed across multiple smaller decisions. Going down one level from the root of the trie, we pay for picking a as the first symbol. This cost is $-log_2$ of the number of strings that start with a divided by the total number of strings in the corpus:

$$-\log_2 \left(\frac{\textbf{Count}(\hat{}a)}{|C|} \right) \tag{6}$$

Once the initial a is fixed, the remaining cost is $-log_2$ of the portion of strings ending with β among strings starting with a:

$$-\log_2 \left(\frac{\textbf{Count}(\hat{}a\beta\#)}{\textbf{Count}(\hat{}a)} \right) \tag{7}$$

The total cost of the string is obtained by adding these two terms:

$$-\log_2 \left(\frac{\textbf{Count}(\hat{}a)}{|C|} \times \frac{\textbf{Count}(\hat{}a\beta\#)}{\textbf{Count}(\hat{}a)} \right) = -\log_2 \left(\frac{\textbf{Count}(\hat{}a\beta\#)}{|C|} \right) \tag{8}$$

This represents the first branching point in the trie, separating the cost of the string into two terms: the cost of the initial symbol and that of the rest of the string. The same reasoning can be applied to the second term for the second and subsequent branching points.

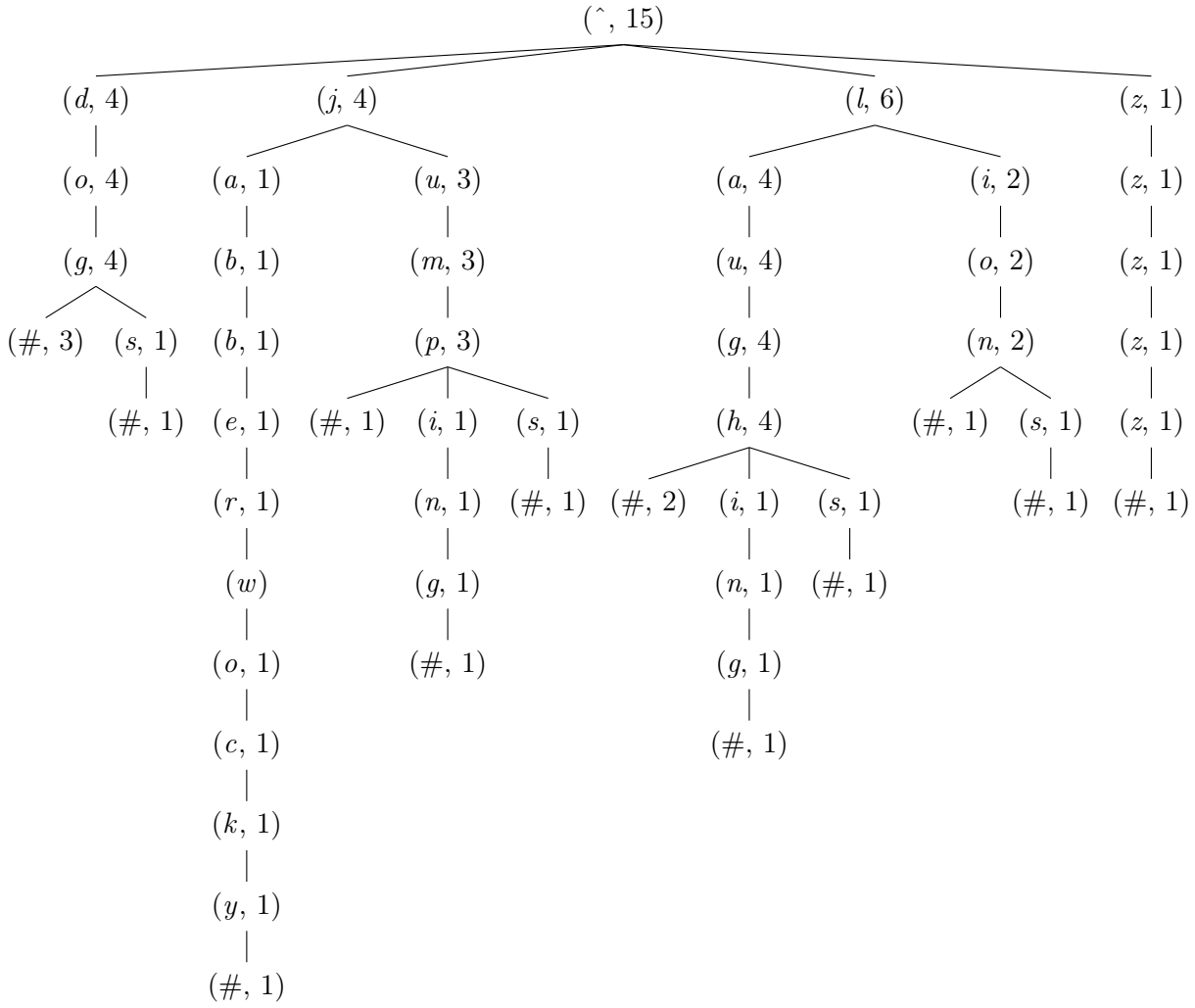

Figure 10: A trie with counts

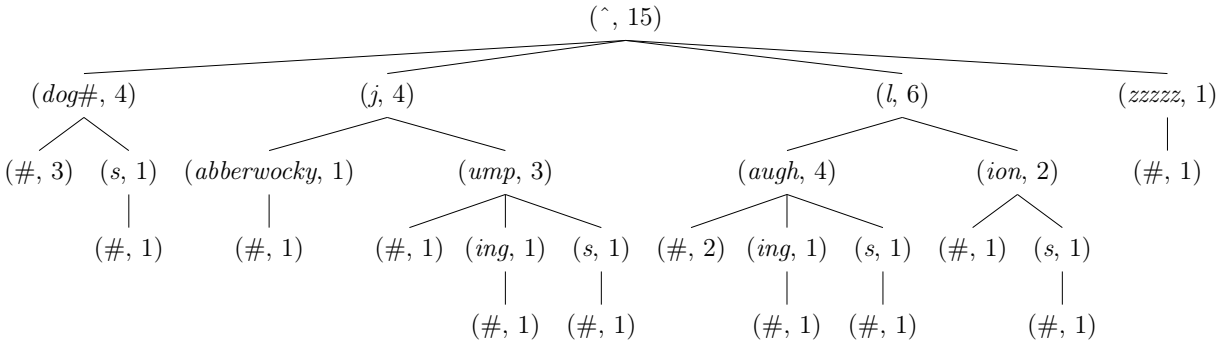

Figure 11: A radix tree with counts

2.4 Linguistica

Linguistica represents a series of grammars that one of us (JG) has developed for learning morphology — or one particular sort of morphology, that of word-internal segmentation (but with relatively little concern for the learning of morphophonology). The generalizations that a linguist can extract for the principles of word-formation in a given natural language amount to redundancies in the word structure, and this redundancy is the information that can be extracted from the words, allowing for a simpler description of any corpus, when by "simpler" we mean specifically a measurement in terms of bits.

Most of the word on *Linguistica* has focused on methods of automatically generating the morphologies from raw data, but that aspect is not of our concern in this paper; our concern here is just on measuring the compression to a corpus based on a description that incorporates morphological structure in a natural fashion.

Using the abbreviation "*lxa*" for Linguistica, we can say that an *lxa*-morphology consists of three things: a list of stems, a list of affixes (in principle divided up, in turn, into a list of prefixes and a list of suffixes), and a list of *signatures*, which is a statement of which stems co-occur with which affixes. A signature could be visualized as in Figure 12, or described as {stem_1, stem_2, ..., stem_m; affix_1, affix_2, ..., affix_n }, a formalism which compresses all of the words that can be generated by combining one of the stems with one of the affixes.[7]

Analyzed (0.206451) :

$S1$ (0.893085) : $\left\{ \begin{array}{l} dog\ (0.584963) \\ lion\ (1.58496) \end{array} \right\}$ $\left\{ \begin{array}{l} NULL\ (0.584963) \\ s\ (1.58496) \end{array} \right\}$

$S2$ (1.11548) : $\left\{ \begin{array}{l} laugh\ (0.807355) \\ jump\ (1.22239) \end{array} \right\}$ $\left\{ \begin{array}{l} NULL\ (1.22239) \\ ing\ (1.80735) \\ s\ (1.80735) \end{array} \right\}$

Unanalyzed (2.90689) :

jabberwocky (1.0)

zzzzz (1.0)

Figure 12: A lxa-grammar

[7] To be sure, prefixes and suffixes work differently, and the distinction has to be attached to each signature. Prefixes precede stems, while suffixes follow stems. Each signature encodes only suffixes or prefixes, never both.

Analyzed words

Word	Signature	Stem	Affix	Flag as analyzed	Count	Sum
dog#	1.11548	0.584963	0.584963	0.206451	3	7.47556
dogs#	1.11548	0.584963	1.58496	0.206451	1	3.49185
jump#	0.893085	1.22239	1.22239	0.206451	1	3.54432
jumping#	0.893085	1.22239	1.80735	0.206451	1	4.12928
jumps#	0.893085	1.22239	1.80735	0.206451	1	4.12928
laugh#	0.893085	0.807355	1.22239	0.206451	2	6.25857
laughing#	0.893085	0.807355	1.80735	0.206451	1	3.71425
laughs#	0.893085	0.807355	1.80735	0.206451	1	3.71425
lion#	1.11548	1.58496	0.584963	0.206451	1	3.49185
lions#	1.11548	1.58496	1.58496	0.206451	1	4.49185

Unanalyzed words

Word	Pointer	Flag as unanalyzed	Count	Sum
jabberwocky#	1	2.90689	1	3.90689
zzzzz#	1	2.90689	1	3.90689
			Total:	52.25484

Table 5: Toy corpus cost using Figure 12

A *lxa*-grammar consists of the following four components, plus a function which is the major component of its UG:

1. a list of stems
2. a list of affixes
3. a list of signatures
4. a list of unanalyzed words (unanalyzed by *Linguistica*, that is).
5. a function whose output is a word, given certain specifications (i.e., the choice of stem and affixes).

The function in question is a parser, taking as its input an lxa-grammar and a word, and outputting a probability if the word is generated by the lxa-grammar. In its implementation, the function looks first to see if the word is an unanalyzed word generated by Linguistica, in which case it simply outputs the product of a stored probability for that word times the probability of an unanalyzed word. In the case of a word analyzed by the lxa-grammar, it finds a stem (from its list of stems) that is a prefix (in the sense of Subsection 2.1) of the word, and from the signature to which that stem belongs, a suffix which completes the word in question. The probability assigned to that analysis is the product of four things: the probability of an analyzed word, the probability of the signature, the probability of the stem given the signature, and the probability of the suffix given the signature. In practice, all of these calculations are done with inverse log probabilities.

2.4.1 Probability distributions

A linguist's grammar, viewed by information theory, is a way to express redundancies in the data, and redundancies are repeated occurrences of "the same thing," in a sense that the linguist must elucidate. A simple case of reuse arises in morphology, where we see that different signatures (i.e., patterns of "permissible" affixation) may involve reuse of the same affix, such as *-s*, *-er* or *-ing* by different signatures.

To express a signature, then, the grammar must have a way to pointing to any particular affix, in the sense that the most common signature in English, for example, is that found on nouns that can occur in the singular (with a null suffix) and the plural (with a suffix *-s*). Other signatures will also want to point to the same suffixes, and each signature will not have to "pay the cost" of spelling out a suffix; the grammar will pay its cost just once, in laying out its suffixes, and the statement of the signature will only have to deal with the *cost*, in bits, of specifying that affix. How many bits does that mean, in actual practice?

This is a question to which information theory provides an unambiguous answer, which is this. The grammar provides a list of potential candidates, and a distribution is provided over that list. A distribution is a set of non-negative (here, positive) numbers that add up to 1.0. In all of the cases associated with *Linguistica* (and all of the other grammars that we explore here as well), these distributions derive from a simple operation: each choice is associated with a positive integer (which comes from counting the use, as we will see) and the distribution is formed by dividing that integer

by the sum of all of these positive integers. In all of the cases we consider, this fraction can be viewed as an empirical frequency. In the case of suffixes, each suffix appears a certain number of times in the corpus, and its frequency is equal to the number of times it has appeared divided by the total number of times any suffix has appeared.

Probability distributions play two very different roles in Linguistica (as they do in any probabilistic grammar). They are involved, first, in specifying the cost of grammatical structure, and second, in indicating the frequency within which certain grammatical decisions are made to generate the data (where the data is the corpus at hand). This difference may seem artificial, but it is both deep and important.

The cost of pointing to an object in a grammar is a plog (i.e., an inverse log probability). The natural unit of a plog is a *bit*. In the current implementation of our project, we "overcharge" a grammar that reports bits in its grammar, because we calculate the bit content of the grammar on the basis of the symbols that it produces in a computer-readable format. Hence, a plog (an inverse base 2 logarithm of a probability) might be represented in the grammar as "4.3364". Rather than treating that as a number of bits, we calculate the cost of that string as 6 symbols times the number of bits required to express a symbol, which is around 6.5, for a total of about 40 bits—as opposed to a little over 4. There are arguments for and against doing this, but the reader should bear in mind that there are a certain number of decisions we have made in the project we report that could have been made differently.

3 Results and discussion

The point of this exercise is to illustrate in a relatively simple example how alternative "linguistic theories" could be evaluated quantitatively with judicious application of MDL analysis. The competition we look at does not pit different theories defended by different linguists—but that is what we would like to do in the future. This paper thus stands as an invitation to other linguists to join us in actually accomplishing this. What follows illustrates the way an evaluation would look.

3.1 Natural language corpora

In order to compare the formalisms, we have selected three natural language corpora:

- English (Brown; Francis and Kucera 1964)
- Swahili (Hurskainen 2004)
- Turkish (Sak et al. 2008)

At the preprocessing stage, each corpus has been truncated to 100000 lines (to ensure roughly similar size), split on whitespaces and line breaks, stripped of some of the punctuation (such as brackets), and converted into a list of words. The total costs for each corpus and formalism, along with a breakdown by term, are presented in Table 6, with the lowest values for each term highlighted.

	UG	**Grammar**	**Corpus**	**Total**
Promiscuous	2,053.871	E: 611.458	E: 37,500,208.281	205,842,583.421
		S: 838.795	S: 108,341,823.028	
		T: 1,669.750	T: 59,995,378.237	
Unigrams	2,759.400	E: 3,966.637	E: 26,159,996.151	132,926,010.728
		S: 5,001.412	S: 69,622,249.822	
		T: 8,105.737	T: 37,123,931.569	
Bigrams	4,452.668	E: 61,749.408	E: 20,958,221.710	102,432,023.980
		S: 114,287.752	S: 52,324,504.528	
		T: 167,523.785	T: 28,801,284.129	
Lists	1,607.037	E: 5,828,118.770	E: 16,747,433.438	110,011,726.765
		S: 12,428,320.721	S: 42,256,599.040	
		T: 12,053,285.372	T: 20,696,362.386	
Tries	7,321.816	E: 5,467,107.757	E: 18,679,448.349	126,824,155.698
		S: 13,622,153.066	S: 51,776,456.268	
		T: 11,261,392.523	T: 26,010,275.917	
Radix trees	7,807.847	E: 3,762,676.713	E: 18,679,448.349	116,453,923.861
		S: 8,657,867.518	S: 51,776,456.268	
		T: 7,559,391.248	T: 26,010,275.917	
Lists with counts	2,555.580	E: 7,187,421.028	E: 11,550,733.734	92,056,572.090
		S: 15,074,616.422	S: 28,198,575.625	
		T: 14,634,907.642	T: 15,407,762.058	
Tries with counts	10,057.698	E: 8,728,969.096	E: 11,550,733.734	102,511,788.735
		S: 20,973,389.936	S: 28,198,575.625	
		T: 17,642,300.588	T: 15,407,762.058	
Radix trees with counts	10,277.196	E: 6,837,745.504	E: 11,550,733.734	90,465,346.987
		S: 14,989,090.709	S: 28,198,575.625	
		T: 13,471,162.162	T: 15,407,762.058	
Linguistica	30,353.397	E: 6,977,659.613	E: 11,678,101.40	98,545,529.052
		S: 17,638,608.323	S: 28,727,758.85	
		T: 18,039,756.059	T: 15,453,291.41	

Table 6: Cross-formalism MDL for natural corpora (E: English, S: Swahili, T: Turkish)

While the purpose of this table is mostly to illustrate the *format* of running a competition between linguistic theories, there are a few observations to be made.

One important takeaway is the relative size of the UG term. These UGs do not grow with the alphabet or corpus size and add only a constant length to the total cost regardless of the corpora. With larger corpora in play, compared to the individual grammars and corpus descriptions, all UGs are negligibly small. This means that technicalities such as variable names and details of implementation are largely irrelevant, what matters is how each linguistic theory deals with the data. The same applies to the choice of a programming language to serve as the UTM: individual grammar costs depend only on their length in symbols and number of distinct symbols used, and corpus costs are produced by the UG parser and the individual grammar.

At the same time, the UG term still plays a crucial role. Just as the grammar term of two-term MDL can be "abused" by including the bulk of the corpus data in it (as the list-based formalisms do), it is in principle possible to have the UG store a lot of corpus-specific data. The presence of the UG term ensures that this strategy, while allowed, does not come for free.

Finally, one of the goals of such a competition is exchange of ideas. The open nature of the comparison allows each participant to examine the standings, broken down by term, as well as other entries; and identify and adopt good strategies used elsewhere. For instance, the next iteration of the competition can have a version of Linguistica that minimizes corpus cost by by recording within signatures, for each affix, its frequency given each stem in the signature — in effect, adopting some of the principles behind radix trees with counts — or straight-up incorporate radix trees for unanalyzed words. Another example is the observation that formalisms that implicitly assume a uniform distribution over symbols or words (promiscuous, lists, tries, radix trees) consistently lose to those making use of counts or explicit probabilities. Learning from this experience, any formalism could be updated to a probabilistic version.

3.2 Future work

From a certain point of view, the evaluation procedure which we draw from MDL analysis resolves itself into a method of comparing encoding algorithms, and that fact is both the surprise lying behind MDL, and potentially a wake-up call for linguists to consider the value of information theory to formal linguistics. At the same time, our project lends itself well to real research questions that are

linguistic in nature. Outlined below are a few classes of problems that could be aided or informed by three-term MDL.

Comparing linguistic theories. As we mentioned earlier, when it comes to ideas in mainstream theoretical linguistics, extended formalization is uncommon. A smattering of formalization here and there does not count for our purposes; often linguists use formalism as a way of expressing their ideas to other linguists, rather than as part of an over-arching formalization of the entire analysis. The entire grammar and linguistic theory must be formalized for the task we have in mind, and while this is normal in computational linguistics, it is not at all normal outside of that discipline. Yet description lengths can only be obtained for linguistic theories that are formalisms — that is, include a clear definition of all components of a grammar, such as its basic entities and rules for putting them together, and a UG that can be expressed as a computer program for a given UTM. For instance, we cannot compare SPE-style ordered rules (Chomsky and Halle 1968) and Optimality Theory (OT) with its ranked constraints (Prince and Smolensky 2008 [1993]) directly. However, we can compare specific formalizations of these linguistic theories — comparing, for example, SPE formalized as finite-state transducers by Kaplan and Kay (1994) against OT formalized as weighted finite-state transducers by Riggle (2004).

The addition of the UG term potentially addresses some objections to standard two-term MDL. For example, Heinz and Idsardi (2013) argue against MDL as "the right basis for generalization" in phonology, illustrating the point with a strictly 2-local, a strictly 2-piecewise, and a counting (regular) pattern. As they point out, if MDL were the appropriate arbiter, then we would expect the subregular classes that describe most phonological phenomena to have shorter descriptions, which is not the case. However, in order for this comparison to be possible at all with two-term MDL, all three patterns must be translated into the same formalism — in this case, finite-state automata, which encode patterns as states and transitions between them. But this approach fails to take into account the difference between formalisms themselves. The simpler subregular grammars, which could be described in terms of allowed or prohibited substrings or subsequences, are effectively forced to pay for machinery that they never use to its full capacity.

Settling debates within a framework. The second use case for cross-formalism MDL is fine-tuning *within* a given framework, comparing variants of the same approach. For instance, OT defines the universal grammar as a set of constraints, whereas individual languages differ in how

these universal constraints are ranked. There are many constraints proposed in the OT literature—no fewer than 1,666 in the period of time between 1993 and 2008, according to (Ashley et al. 2010), and debate exists on which of them should be accepted or rejected.[8] Cross-formalism MDL would be able to compare these versions of (some formalization of) OT; from its point of view, the differences boil down to which constraints are part of the UG term, and which (if any) are encoded within individual language-specific grammars.

Another example, in the domain of syntax, is provided by minimalist grammars. Devised by Stabler (1997) as a formalization of Chomsky's 1995; 2000 Minimalist Program, this formalism has spawned a multitude of variants and extensions. Examples include minimalist grammars augmented with ellipsis (Kobele 2012); adjunction and hierarchy of projections (Fowlie 2013); covert movement, coordination, and across-the-board movement (Torr and Stabler 2016); agreement (Ermolaeva and Kobele 2023); and others. Cross-formalism MDL offers a way to determine which additions are "worth it" in an explicit way, and by how many bits each addition makes itself worth it.

Grammar modules. As observed in Section 2, some of the baseline formalisms encode phonotactic patterns, while others can be considered theories of morphology. A linguist would typically want to capture both as distinct modules of grammar. Translating this into cross-formalism MDL, we would have to pay the UG cost of two separate formalisms as well as effectively maintain two grammars per corpus. The question in this case is whether the extra cost would be outweighed by better compression of the corpus. An example would be a version of Linguistica for morphology used in conjunction with a simple unigram or bigram grammar for phonology.

Comparing formalizations. Finally, different formalizations of the same mainstream linguistic theory can themselves be evaluated using cross-formalism MDL. An example is once again provided by OT, which has been given a number of formal treatments including, among others, various finite-state approaches such as (Karttunen 1998; Gerdemann 2000; Riggle 2004). Returning to the Minimalist Program for syntax, another interesting case is presented by minimalist grammars vs. the more recent (and more faithful to the mainstream) formal account in (Collins and Stabler 2016).

[8] The suggestion has even been made (in (Cho 2003)) that some constraints may be language-specific, and that opens up a number of cans of worms that would bear on the complexity of the UGs across different implementations of OT.

3.3 Final remarks

3.3.1 Large language models

Like some but by no means all linguists, we believe that the successes of large language models (LLMs) obliges linguists to think carefully about the nature of modern grammatical analysis, and just where it is that grammatical theory can justify its existence in a world in which tasks which had seemed beyond the reach of linguistics (such as automatic translation) are handled well by systems that may or may not contain within them something that resembles grammatical analysis. The project that we describe in this paper may be useful in coming to grips with these questions to some degree.

It is not only the size of the corpus that distinguishes LLMs from traditional grammatical analyses, though size is the most obvious distinguishing characteristic; there is also a heavy dependency of the analysis on the corpus used for analysis, and a sense that as the corpus gets much larger, by orders of magnitude, the nature of the problems that can be solved slowly morph into larger tasks. Traditional grammatical analysis gives no reason to think this, while MDL-based analysis shows us explicitly that as the data get larger, more finely articulated parts of a grammar become "worth it" in a sense we have already alluded to. We mentioned earlier that it is common in real world computational projects to include such categories as days of the week, and months of the year. The categories of seven days words and of 12 month words have properties that distinguish themselves from many other words of the same frequency, and setting up such categories simplifies the grammar from an MDL perspective, even though, of course, it *adds* a certain amount of complexity to the grammar. MDL motivates the behavior of human computational linguists in creating more lexical categories as the corpora being considered grow large.

LLM analysis also worry about over-fitting the corpora they analyze, and the term used in machine learning (more generally) for methods that deal with that problem is *regularization*: regularization prunes complexities that an automatic learning process has come up with that are not justified by how they generalize to unseen data. That concern is very much the same as the one that MDL tries to capture: when is an addition to the grammar justified by the generalizations seen so far in the data? That question motivates both LLM modelers and grammarians, in our view. MDL

is one possible bridge that would allow researchers in these two areas to find a common language and some common ground.

3.3.2 Is MDL enough?

We would like to discuss briefly two points that seem important to us about the use of MDL analysis in developing linguistic grammars and thinking about the nature of linguistics. The first involves the relative importance of the terms "minimal" and "description length" in the term "minimal description length," while the second involves the way in which linguists rethink the use of the term "redundancy" as it is used in information theory.

As MDL is often described, its greatest strength lies in its ability to give a principled reason for deciding when analysis of data should stop, at pain of overfitting the data at hand. That is a noble goal, and if MDL succeeds there, that is great. But in many cases we have worked on, this aspect of MDL is not all that we have hoped it to be. And yet MDL's potential contribution to our work does not seem to vanish. Why is that? The reason, it seems to us, is that MDL analysis brings with it something that is present long before the analysis decides it should stop because a minimum value of description length has been achieved.

To create an MDL-stype analysis, the linguist is forced to think about how the expression of a robust grammatical generalization can be formulated so that the generalization's presence is no longer *in* the data, but rather removed from the data itself. When things are done correctly, what results from that is a generalization that takes relatively few bits to express, but the data have shrunk by a much bigger amount by virtue of the extraction of that generalization. This style of working forces the linguist to look at what is present in the data so that emergent generalizations can be extracted. In the case of Linguistica, which has played a role in this paper as an exemplar of how a grammar could use MDL and reduce description length, the grammar extracts (for example) the simple fact that the affixes described here are all suffixes. A simple fact, to be sure, but extracting that generalization removes a bit from the analysis of each morphologically complex word. It does much more, of course; it allows stems to profit from the fact that their patterns of permitted affixes are shared by a large number of kindred stems, again saving bits in the data by extracting a generalization.

It seems to us that it is possible — very possible — that MDL can play a very healthy role in grammar development, in the way we have laid out here (and in Goldsmith 2015) without committing ourselves to the idea that the absolutely shortest description length is the one that we linguists want. For example, it could well be the case that the grammar induction algorithms stop short of compressing as well as theoretically possible (as short as a good compression algorithm could produce, let us say)—but that might be just what the linguist wants. And it may still be true that comparing linguists' models by comparing the description lengths produced will give us deep insight into which grammar is the best, from a linguistic point of view.

The second point we would like to draw out involves an account of why and how information theory and linguistics failed to enter into a healthy conversation early in the development of information theory. A good part of the reason, it seems to us, lies in the unfortunately choice of the word "redundancy" in information theory. In information theory, it is often suggested that one can reduce the apparent information in a message by extracting redundancy (or redundancies), and that sounds like a good thing—as if something costly were being wasted when it did not need to be wasted. But that is a bad metaphor, and for the analysis of natural language, it is a terrible metaphor. The redundancy which the linguist extracts is nothing more or less than the linguistic structure: if it is redundant for the information theorist, it is pure gold for the linguist. To put it as an apothegm: redundancy *is* structure, and structure is redundancy. If Shannon and Jakobson could have agreed on that, we might have moved forward more quickly in the 1950s.

References

K. C. Ashley, L. Disch, D. C. Ford, E. MacSaveny, S. Parker, C. Unseth, A. M. Williams, R. Wong, and B. Yoder. How many constraints are there? a preliminary inventory of ot phonological constraints. *Occasional Papers in Applied Linguistics*, 9:1079–0610, 2010.

G. J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13(4):547–569, 1966.

H.-S. Cho. Non-universality in outimality theory: An information-based model. *SNU Working Papers in English Language and Linguistics*, 2, 2003.

N. Chomsky. *Aspects of the theory of syntax*. MIT Press, Cambridge, MA, 1965.

N. Chomsky. *The Logical Structure of Linguistic Theory*. Springer US, 1975 [1955].

N. Chomsky. *The Minimalist Program*. MIT Press, Cambridge, MA, 1995.

N. Chomsky. Minimalist Inquiries: the framework. In R. Martin, D. Michaels, and J. Uriagereka, editors, *Step by Step: Essays on Minimalist Syntax in Honor of Howard Lasnik*, pages 89–156. MIT Press, Cambridge, MA, 2000.

N. Chomsky and M. Halle. *The sound pattern of English*. Harper & Row, New York, NY, 1968.

N. Chomsky and H. Lasnik. Filters and control. *Linguistic Inquiry*, 8(3):425–504, 1977.

C. Collins and E. Stabler. A formalization of minimalist syntax. *Syntax*, 19(1):43–78, 2016.

C. de Marcken. Unsupervised language acquisition. arXiv preprint cmp-lg/9611002, 1996.

M. Ermolaeva and G. M. Kobele. Formal properties of agreeing minimalist grammars. *Proceedings of the Society for Computation in Linguistics*, 6(1):385–388, 2023.

M. Fowlie. Order and optionality: Minimalist grammars with adjunction. In *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 12–20, Sofia, Bulgaria, 2013. Association for Computational Linguistics.

W. N. Francis and H. Kucera. A standard corpus of present-day edited American English, for use with digital computers. *Brown University, Providence*, 2, 1964.

E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

D. Gerdemann. Approximation and exactness in finite state optimality theory. In *Proceedings of the Fifth Workshop of the ACL Special Interest Group in Computational Phonology*, pages 34–45. University of Luxembourg, 2000.

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

J. Goldsmith. An algorithm for the unsupervised learning of morphology. *Natural Language Engineering*, 12(4):353–372, 2006.

J. Goldsmith. Towards a new empiricism for linguistics. In *Empiricism and Language Learnability*, pages 58–105. Oxford University Press, Oxford, UK, 2015.

P. D. Grünwald. *The minimum description length principle*. MIT Press, Cambridge, MA, 2007.

J. Heinz and W. Idsardi. What complexity differences reveal about domains in language. *Topics in cognitive science*, 5(1):111–131, 2013.

Y. Hu, I. Matveeva, J. Goldsmith, and C. Sprague. Using morphology and syntax together in unsupervised learning. In *Proceedings of the Workshop on Psychocomputational Models of Human Language Acquisition*, pages 20–27, Ann Arbor, MI, 2005. Association for Computational Linguistics.

A. Hurskainen. Helsinki corpus of Swahili. *Compilers: Institute for Asian and African Studies (University of Helsinki) and CSC*, 2004.

R. M. Kaplan and M. Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

L. Karttunen. The proper treatment of optimality in computational phonology. In *Finite State Methods in Natural Language Processing*, 1998.

G. M. Kobele. Eliding the derivation: a minimalist formalization of ellipsis. In *Proceedings of the HPSG 2012 conference*, 2012.

A. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of information transmission [Problemy peredachi informatsii]*, pages 3–11, 1965.

J. Lee and J. 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: Demonstrations*, pages 22–26, 2016.

D. R. Morrison. Patricia—practical algorithm to retrieve information coded in alphanumeric. *J. ACM*, 15(4):514–534, oct 1968. ISSN 0004-5411. doi: 10.1145/321479.321481.

A. Prince and P. Smolensky. *Optimality Theory: Constraint Interaction in Generative Grammar*. Wiley, 2008 [1993]. doi: 10.1002/9780470759400.

E. Rasin and R. Katzir. On evaluation metrics in optimality theory. *Linguistic Inquiry*, 47(2): 235–282, 2016.

E. Rasin, I. Berger, N. Lan, and R. Katzir. Learning phonological optionality and opacity from distributional evidence. In S. Hucklebridge and M. Nelson, editors, *Proceedings of the North East Linguistic Society 48 (NELS 48)*, volume 48, pages 269–282, Amherst, MA, 2018.

J. A. Riggle. *Generation, recognition, and learning in finite state Optimality Theory*. PhD thesis, University of California, Los Angeles, 2004.

J. Rissanen. *Stochastic Complexity in Statistical Inquiry*. Computer Science Series. World Scientific Publishing Company Pte Limited, 1989.

H. Sak, T. Güngör, and M. Saraçlar. Turkish language resources: Morphological parser, morphological disambiguator and web corpus. In *Advances in Natural Language Processing: 6th International Conference, GoTAL 2008 Gothenburg, Sweden, August 25-27, 2008 Proceedings*, pages 417–427. Springer, 2008.

R. J. Solomonoff. A formal theory of inductive inference, parts I and II. *Information and Control*, 7 (1 & 2):1–22, 224–254, 1964.

E. P. Stabler. Derivational minimalism. In C. Retoré, editor, *Logical Aspects of Computational Linguistics: First International Conference, LACL '96 Nancy, France, September 23–25, 1996 Selected Papers*, pages 68–95. Springer, Berlin, Germany, 1997.

J. Torr and E. Stabler. Coordination in minimalist grammars: Excorporation and across the board (head) movement. In D. Chiang and A. Koller, editors, *Proceedings of the 12th International Workshop on Tree Adjoining Grammars and Related Formalisms (TAG+ 12)*, pages 1–17, Düsseldorf, Germany, 2016.

J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.