A Minimum Description Length Approach to Grammar Inference

Peter Grünwald*

CWI, P.O. Box 4079, 1009 AB AMSTERDAM, The Netherlands

Abstract. We describe a new abstract model for the computational learning of grammars. The model deals with a learning process in which an algorithm is given an input of a large set of training sentences that belong to some unknown grammar. The algorithm then tries to infer this grammar. Our model is based on the well-known *Minimum Description Length Principle*. It is quite close to, but more general than several other existing approaches. We have shown that one of these approaches (based on n-gram statistics) coincides exactly with a restricted version of our own model. We have used a restricted version of the algorithm implied by the model to find classes of related words in natural language texts. It turns out that for this task, which can be seen as a 'degenerate' case of grammar learning, our approach gives quite good results. As opposed to many other approaches, it also provides a clear 'stopping criterion' indicating at what point the learning process should stop.

1 Introduction

In this paper we construct an algorithm that has as its sole input a *training text*, that is, a very large fragment of a natural language text in for example English. It should then output grammatical rules underlying this language; so in this case, English grammar rules. Thus learning takes place using positive examples only – possibly including noise. No explicit negative example is ever given. Approaches along these lines are commonly called [8] 'Grammatical Inference'. It should immediately be noted that there are severe limits to the grammatical inference approach: we will simply use context-free grammars and the fact that they are not capable of fully capturing natural language grammars is just one of many weaknesses of the approach [8]. Nevertheless, it will turn out to be interesting to see just how far one can get with such an abstract approach.

Our algorithm is based on the *Minimum Description Length* (MDL) *Principle*, introduced in its present form by Rissanen [10]. The original ideas behind this very general principle stem from R.J. Solomonoff. In his landmark 1964 paper [11] they were already applied to grammar inference. In the next two sections we will give a very short introduction to the MDL Principle and to how it can be

* E-mail: pdg@cwi.nl. Partially supported by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556, and by NWO through NFI Project ALADDIN under Contract number NF 62-376.

used for grammar induction. Section 4 discusses our approach, first describing its algorithmic aspects and then explaining how we compute description lengths. The reader who wants to see first what this approach leads to before worrying about the details may want to have a look at Sect. 5 first, where we discuss the results of some experimental data we obtained with an implementation of our algorithm. Section 6 compares our model to some others and discusses whether our approach makes sense from a linguist's point of view. The paper ends with some concluding remarks.

2 The MDL Principle

One of the basic ideas of Solomonoff's theory of induction[2] [11] is that 'learning' can, under the right circumstances, be seen as 'finding a shorter description of the observed data'. Here one views learning as finding a hypothesis that explains some observed data and makes predictions about data yet unseen. The MDL Principle, introduced in its present form by J. Rissanen [10], is a principle of statistics that is explicitly based upon Solomonoff's ideas. It goes as follows:

Minimum Description Length Principle. *The best theory to explain a set of data is the one which minimizes the sum of*

1. *the length, in bits, of the description of the theory;*
2. *the length, in bits, of the data when encoded with the help of the theory.*

The general idea here is that an 'empty' theory (the words 'theory' and 'hypothesis' will be used interchangeably here) which does not give any clue as to the sort of data we are dealing with, has no predictive value whatsoever. On the other hand, a theory that describes all of the data perfectly is very much restricted by the data and therefore does not really generalize. Therefore, it has no predictive value either. We have to find some trade-off, and in the next section we will see in a practical setting that the total description length can become much smaller using such a trade-off.

3 Grammar Learning and the MDL Principle

Consider three grammars that can account for a given natural language text [11]:

the promiscuous grammar This is an 'empty' grammar: given an input alphabet (vocabulary) of words, it accepts every concatenation of them as a sentence. Of course, it is a hopeless overgeneralization.

the ad hoc grammar This grammar accepts exactly all the sentences it has seen so far, but no single different sentence. Of course, it is a hopeless undergeneralization.

[2] See [9] for a good introduction to Solomonoff's theory.

the 'right' grammar

This is just a context-free grammar of English as can be found in several textbooks. In a sense, it is the best generalization we can arrive at within the context free grammar framework.

Now consider the description length we arrive at if we want to reproduce the entire training text of length (i.e. the number of words) L, where we may choose L to be much larger than the description length of the right grammar. From the MDL point of view, here the training text fulfills the role of the *data*, and the grammar the role of the *theory*:

The promiscuous grammar (which has a very short description size of course) says nothing at all about the example sentences that have been seen so far; so in order to reproduce these sentences, we must list them all explicitly. The ad hoc grammar (which has a description size of about L) completely describes the training text (provided we have written down and numbered the production rules $S \rightarrow <$ `sentence` $>$ in the same order as in the training text). The ad hoc grammar thus provides a data description of size approximately 0 but a grammar description of size approximately L. The total description lengths for both the ad hoc and the promiscuous grammar are of size about L. The right grammar however makes for a much smaller description of the data, while it has a fixed theory size. Therefore its total description length will be much lower than L, provided our training text is long enough. To see why this is so, one must appreciate the *constraining* function of a grammar:

Suppose the first word of a sentence has turned out to be an interrogative pronoun (like *where*, *who*, etc.). The grammar now predicts that the next word can *only* be a verb[3]! If we suppose that about $1/4$ of all different words in natural language are verbs, this means that, given a vocabulary of size n, we need to reserve about $\log n - 2$ bits for this word. This is 2 less than the $\log n$ we would need for the unique specification of this word without the help of a grammar.

Each regularity of this kind helps to reduce the description length for some words in the training text. Therefore, the longer the training text, the bigger the savings will be. We have made use of regularity in the training text in order to shorten its description. The shortest description possible is the one that makes use of all the regularity that may be found at all – and for large training texts this can only be a grammar similar to the one we were looking for [11]!

Note that we have now found a *criterion* to find out if learning has occurred (we arrive at a shorter description length of theory+data given the theory), but we have not yet given any guidelines as to *how* learning should take place. Trying to keep things as abstract as possible, we will use the description length for that, too. Simply trying out all grammars that are consistent with a given text and

[3] Of course, this is not entirely true ('What *the* ... does this research lead to??'). But we will see in section 6 that the fact that the next word will be a verb *with high probability* can already be used to reduce the description length of the training text.

picking the one that yields the shortest description is infeasible. Therefore we adopted a 'greedy' approach: One starts with a set of possible hypotheses, the *hypothesis space* $\mathcal{H}$ (consisting of all grammars that could account for a given training text. E.g., the set of all context free grammars). One picks one of the hypotheses $h \in \mathcal{H}$ as the current one, and one has *operators* transforming the current hypothesis into another hypothesis $h' \in \mathcal{H}$. At each step, one performs the operation that yields the highest immediate decrease in description length (of the new hypothesis plus the data given this new hypothesis). If no such operation is possible any more, learning stops, and the current hypothesis is output as the 'best' one. Of course one may get stuck in 'local minima' using this approach. In the following section we look at the choices that have to be made and the details that have to be filled in if one wants to turn such a greedy approach into a *practical* model.

4 Our Approach

The Algorithm

The input to our algorithm is a training text $t_1^L = t_1 t_2 \ldots t_L$. L is the total length of the training text; t_i stands for the i-th word of the training text. We have an alphabet $\mathcal{W}$ of different (distinct) words, and we will index the elements w of $\mathcal{W}$ from 1 to $|\mathcal{W}|$. So for each i, $1 \leq i \leq L$, there is exactly one j, $1 \leq j \leq |\mathcal{W}|$, such that $t_i = w_j$.

Given t_1^L, the algorithm tries to build a grammar covering it. It does so in a bottom-up way. We define a *partial grammar* $\mathcal{C}$ as a set of context-free grammar rules for a training text, but not necessarily with a starting symbol. Thus correct sentences may be partially parsed, but sometimes not be completely reduced to the non-terminal *Sentence*. In [6] we make the notion of 'partially parsing' precise. One can describe $\mathcal{C}$ in a somewhat non-standard way as a set of classes $c \in \mathcal{C}$. For each class c there is exactly one corresponding 'non-terminal' c'. The class then is the set of production rules with this non-terminal on the left-hand-side. There are two basic operations: the merge operation or *merge* for short and the construction operation or *construct* for short.

If two productions c_l and c_k are merged, then they disappear from $\mathcal{C}$ and are replaced by a new class c_n that contains the union of the grammar rules in c_l and c_k, with all left-hand sides replaced by c'_n. For example, suppose for the moment we have arrived at a $\mathcal{C}$ with $|\mathcal{C}| = 10$ and $c_5 = \{c'_5 \rightarrow c'_6 c'_7 | c'_2\}$ and $c_4 = \{c'_4 \rightarrow c'_3\}$. Now if we merged c_5 and c_4, we would arrive at a new class $c_{new} = \{c'_{new} \rightarrow c'_6 c'_7 | c'_2 | c'_3\}$ and we would remove c_5 and c_4 from $\mathcal{C}$ and then re-index $\mathcal{C}$ so as to incorporate c_{new}. Because it simplifies the notation, we will often write this step as if we would let the merged classes continue to exist and simply increase the number of classes by one. In our example, this means that we would introduce a new class $c_{11} = \{c'_{11} \rightarrow c'_5 | c'_4\}$. Notice that this yields an equivalent new partial grammar; but we will see later that when computing the description length, it is important to keep in mind that a merge really *decreases* the number of classes by one in stead of increasing them, as the simpler notation wrongly suggests.

If two classes c_l and c_k are constructed, then a new class c_m is formed containing just one production rule: $c_m = \{c'_m \rightarrow c'_l c'_k\}$. The basic functioning of our algorithm is shown in Fig. 1.

1 **ASSIGN** ONE SEPARATE CLASS TO EACH DIFFERENT WORD THAT OCCURS IN THE TRAINING TEXT: $c_1 := \{c'_1 \rightarrow w_1\}$, $c_2 := \{c'_2 \rightarrow w_2\}$ ETC. TOGETHER THESE CLASSES FORM THE INITIAL PARTIAL GRAMMAR $\mathcal{C}$.
2 **COMPUTE** THE TOTAL DESCRIPTION LENGTH OF THE TRAINING TEXT WHICH IS THE SUM OF THE DESCRIPTION LENGTH OF THE TRAINING TEXT *given* THE SET OF CLASSES $\mathcal{C}$ AND THE DESCRIPTION LENGTH OF THE SET OF CLASSES $\mathcal{C}$ ITSELF.
3 **COMPUTE** FOR ALL PAIRS OF CLASSES $c_i, c_j \in \mathcal{C}$: THE *difference* IN DESCRIPTION LENGTH THAT WOULD RESULT FROM A MERGE OF THESE TWO CLASSES RESULTING IN A NEW CLASS $c_n = \{c'_n \rightarrow c'_i | c'_j\}$.
4 **COMPUTE** FOR ALL PAIRS OF CLASSES $c_i, c_j \in \mathcal{C}$: THE *difference* IN DESCRIPTION LENGTH THAT WOULD RESULT FROM A CONSTRUCTION OF A NEW CLASS $c_m = \{c'_m \rightarrow c'_i c'_j\}$.
5 **IF** THERE ARE ONE OR MORE OPERATIONS FOR WHICH THE NEW DESCRIPTION LENGTH WOULD BE SMALLER THAN THE OLD ONE, **PERFORM** THE OPERATION THAT YIELDS THE SHORTEST NEW DESCRIPTION LENGTH AND **CONTINUE** AT STEP 2.
6 **ELSE** STOP THE EXECUTION.

Fig. 1. the basic algorithm

How to Compute the Description Length

We must come up with a method that defines a description length DL in an exact manner for all possible training texts and all possible grammars (i.e. all possible $\mathcal{C}$). We can do this using a *code* that defines a unique representation for each different training text. We want our code to be *dense*[4], i.e. each sequence of codewords defines a training text t_1^L and no two different sequences of codewords define the same t_1^L. The idea is now that we use our partial grammar $\mathcal{C}$ in such a way that it can be seen as such a code; but information theory [5] teaches

[4] If we do not use a dense code, the nice theoretical result of the equivalence between a restricted version of our approach and another one (Sect. 6), does not hold any more.

us that in order to arrive at an ideal code (i.e. a code that yields the shortest possible representation (description) of the training text) we need to keep track also of the frequencies of occurrence of the classes in $\mathcal{C}$ for a given t_1^L.

Furthermore, in order to make the construct and merge operations really capable of reducing the DL, we must somehow incorporate in our code the basic fact about grammars discussed before: the earlier words (classes) in a sentence (parse tree) partially determine the words (classes) that may follow. Only if our code can make use of these dependencies, can construct and merge operations really reduce the DL.

In our approach, we let each class (i.e. non-terminal) determine a *distribution* on all classes that may be its direct successors. By using a distribution instead of some hard rule of the form 'c_j may follow c_i; c_k may not' we automatically handle noise in the training text. For a class c_i we arrive at the distribution of its successors by counting, for all j, $1 \leq j \leq |\mathcal{C}|$ the number of times we encounter that class c_i turns out to be followed by class c_j while parsing t_1^L according to our grammar $\mathcal{C}$. This determines a set of parameters χ_{ij}. Notice that if we normalized χ_{ij} by dividing it by a constant for all j (keeping i fixed), such that $\sum_{1 \leq j \leq |\mathcal{C}|} \chi_{ij} = 1$, then we could regard the set of all χ_{ij} as a crude estimate of a kind of probability distribution. If our training text were representative of its underlying language, then we might view this distribution as giving the conditional probability that, given that at some point in some parse tree of some natural language text of the same language as t_1^L, we have encountered non-terminal c'_i, the next non-terminal we encounter will be c'_j.

Two things should be noted here. First, we use only dependencies between one (non-) terminal and its immediate successor here, thereby making our approach related to those using 2-gram statistics [1]; we will see however in Sect. 6 that, at least theoretically, our approach can very well handle long-term dependencies. Second, the 'number of times c_j follows c_i while parsing t_1^L according to $\mathcal{C}(t_1^L)$' and thus the set of parameters χ_{ij}, is only well-defined if $\mathcal{C}$ is *unambiguous* while of course, interesting grammars may very well be ambiguous; we will further discuss this issue in Sect. 7.

Actually Computing the Description Length

More formally, we define a *2-gram stochastic partial grammar* for a training text t_1^L as a tuple:

$$\mathcal{G} = < \mathcal{C}, \mathcal{P}_1, \mathcal{P}_2 > \quad . \tag{1}$$

Both $\mathcal{P}_1$ and $\mathcal{P}_2$ are sets of parameters. They are dependent on both t_1^L and $\mathcal{C}$, so they may also be seen as functions of them. $\mathcal{C}$ is a partial grammar as defined before. Notice that $\mathcal{C}$ defines a *shortest possible parse* of t_1^L which we will write as $\mathcal{C}(t_1^L)$. This means that if one parses t_1^L according to $\mathcal{C}$ in a bottom-up fashion, one arrives at a string of classes $u_1^{L'}$ where $L' \leq L$. If $\mathcal{C}$ is unambiguous, as we assume here, then it determines a *unique* way to parse t_1^L into a string that is as short as possible, so $\mathcal{C}(t_1^L)$ is well-defined. Due to restrictions on the size of this chapter, we will have to use words in the definitions of $\mathcal{P}_1$ and $\mathcal{P}_2$. It should be noted that if one introduces the right notations, one can express all these definitions in a purely mathematical fashion [6]. In order to define $\mathcal{P}_1$ and $\mathcal{P}_2$, we first have to define their contents (we have to be really precise now. We used χ_{ij} to explain the general idea, but it turns out that we cannot use just one kind of parameter here - we need to make a distinction between two quite different sorts of parameters θ_{ij} and ν_{ij}):

$$\begin{aligned}
\theta_{ij} &= \textit{number of times } c_j \textit{ follows } c_i \textit{ in } \mathcal{C}(t_1^L) \ . \\
\nu_{ij} &= \textit{number of times production } c_i' \rightarrow \ldots \mid \ldots \mid c_j' \mid \ldots \mid \ldots \\
&\quad\ \textit{is used to replace } c_j' \textit{ by } c_i' \textit{ in the bottom-up parsing process} \\
&\quad\ \textit{that reduces } t_1^L \textit{ to } \mathcal{C}(t_1^L) \ . \\
\mathcal{P}_1 &= \{\theta_{ij} \mid c_i c_j \textit{appears in } \mathcal{C}(t_1^L)\} \ . \\
\mathcal{P}_2 &= \{\nu_{ij} \mid \{c_i' \rightarrow \ldots \mid \ldots \mid c_j' \mid \ldots \mid \ldots\} \in \mathcal{C}\} \ .
\end{aligned}$$

To turn $\mathcal{G}$ into a code (description) of t_1^L, we now turn to information theory [5]. Each $\mathcal{G}$ determines an ideal (Shannon-Fano) code to encode a training text. For our purposes, it is not necessary to explicitly construct this code; in stead, it is enough only to compute the number of bits $DL(t_1^L \mid \mathcal{G})$ it would take to encode training text t_1^L using this ideal code for $\mathcal{G}$. It should be clear that the two terms of the right-hand side of the following equation provide just the information needed to encode t_1^L in such a way that it can be completely reproduced:

$$\begin{aligned}
DL(t_1^L \mid \mathcal{G}) &= \textit{number of bits it takes to encode at all positions in the string} \\
&\quad\ \mathcal{C}(t_1^L) \textit{ what class is the next one} \\
&\quad + \textit{the number of bits it takes to encode all choices that must} \\
&\quad\ \textit{be made on which production to apply when generating } t_1^L \\
&\quad\ \textit{back from } \mathcal{C}(t_1^L) \\
&= \mathbf{H_1}(\mathcal{P}_1) + \mathbf{H_2}(\mathcal{P}_2) \ . \qquad (2)
\end{aligned}$$

$\mathbf{H_1}$ and $\mathbf{H_2}$ can be seen as entropy-like functions on $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively. For explicit expressions, the reader is referred to [6]. Suffice it to say here that to arrive at an expression for the first term, one should realize that $\mathcal{P}_1$ can be viewed as describing the parameters of a stationary ergodic Markov Chain; after some normalizing, the definitions of the entropy of such a chain [5] can be used for quick computation of the first term of (2). Regarding the second term, we may interpret each class as a 'source' (in the sense of information theory) that can emit any of the productions in the class as outputs. $\mathcal{P}_2$ determines the entropy of this source (class). From the set of entropies of all classes, one can easily compute the second term of (2).

The total description length is of course given by

$$DL(t_1^L) = DL(t_1^L \mid \mathcal{G}) + DL(\mathcal{G}) \ .$$

We have just computed the first term. Now for the second. From (1) we see that we can split this term up as follows:

$$DL(\mathcal{G}) = DL(\mathcal{C}) + DL(\mathcal{P}_1) + DL(\mathcal{P}_2) \ .$$

$\mathcal{C}$ is a set of grammar rules. Notice that for a natural language this should have some fixed length. On the other hand, we can take L about as large as we want. Therefore, from some L onwards, $DL(\mathcal{C})$ will be neglectably small as compared to $DL(t_1^L \mid \mathcal{G})$, as this latter term does grow with increasing L. For this reason, we will neglect $DL(\mathcal{C})$ in our computation.

On the other hand, $DL(\mathcal{P}_1)$ and $DL(\mathcal{P}_2)$ *do* grow as our training text increases in size. To see why this is so, one should realize that the *range* of each θ_{ij} increases as L increases (theoretically θ_{ij} can range from 0 to $L-1$). Therefore, the number of bits required to specify all θ_{ij} increases too; it takes about $O(\log L)$ bits. The same reasoning holds for $DL(\mathcal{P}_2)$. We will omit the exact definitions for $DL(\mathcal{P}_1)$ and $DL(\mathcal{P}_2)$ here, but it should be clear by now that they can easily be constructed.

5 Some Results

We have made an initial implementation of the algorithm just described and let it run on the 'Brown Corpus' [7], a huge sample of rather modern American English. In order for the algorithm to work within the limits of available computer power, some adjustments had to be made; details about this can be found in [6][5]. An important one is the fact that we used as training text not the full Brown corpus, but the reduction of the Brown corpus to just those sentences *all* words of which are among the 10000 most frequent words in the corpus. We named our reduced training text 'Brown 10000'. Also, classes of the form $\{c'_i \to w_i\}$ were allowed to be the operands of merges and constructs only if w_i was among the 450 most frequent words.

Just Merging

First, we will discuss the results of running a restricted version of our algorithm, namely one in which only merges and no constructs are allowed. In this case, the algorithm yields a $\mathcal{C}$ each element of which is of the form $c_n = \{c'_n \to w_i|w_j|w_k|\ldots\}$; as c'_n is used in no right-hand side of any production, one may view a class simply as a set of words now. Then 'grammar inference' degenerates into finding classes (hence the name!) of syntactically and/or semantically related words, a task for which automatic procedures have often been proposed before [1, 4].

Figures 2, 3 and 4 show the results of running this algorithm on 'Brown 10000'. The first figure shows the total description length of the training text after each merge. One can see that in the beginning it goes down quite rapidly, each merge clearly giving a more compact description of the training text. Then, at some point (after about 160 merges), the merge operation does not really 'help' any more and the description length starts to rise again. If one does not

[5] The algorithm discussed there differs somewhat from the one we present here, but the necessary adjustments are the same.

stop the algorithm, it continues until all 450 words are in one class and the DL is *very* high (we stopped after 310 merges). Figure 3 gives a more fine-grained picture showing the three terms that sum to $DL(t_1^L)$: $DL(\mathcal{G})$, $\mathbf{H_1}(\mathcal{P}_1)$ and $\mathbf{H_2}(\mathcal{P}_2)$. This decomposition clarifies what exactly goes on here: as more merges take place, the number of parameters decreases: if we only allow merges, the *effective* number of classes decreases by one after each merge, so there are less θ_{ij} to store. Then describing $\mathcal{P}_1$ takes less bits and $DL(\mathcal{G})$ decreases. As for explaining the form of the other two curves, we refer to [6] where we prove a theorem that $\Delta\mathbf{H_2}(\mathcal{P}_2) \geq -\Delta\mathbf{H_1}(\mathcal{P}_1)$ for each possible $\mathcal{C}$.

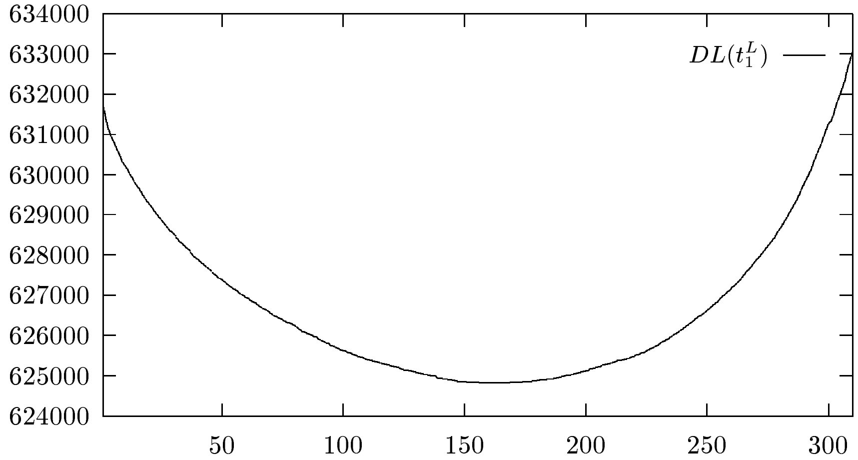

Fig. 2. Brown 10000: Description Length after each merge

Figure 4 focuses on the 'best' situation: it shows so-called *dendrograms* [4] of a (representative) part of the class structure that has been arrived at after 160 merges, when the description length is smallest. In a dendrogram a vertical link between two words means that they have been merged together, the two horizontal lines that start out from the middle of a vertical line and end into the two edges of another vertical line more to the left indicate a merge between two classes. The farther to the left, the higher the number of steps the algorithm has already performed. Words that are not connected are not in the same class. We would say that the results speak for themselves.

Grammar Rules

If we do allow constructs to happen, there arises a problem in computing ΔDL, the difference in description length that would result from a construct of two classes c_i and c_j. We would have to compute this difference for all pairs of classes in $\mathcal{C}$ and this simply takes too much time. In our current implementation, we used

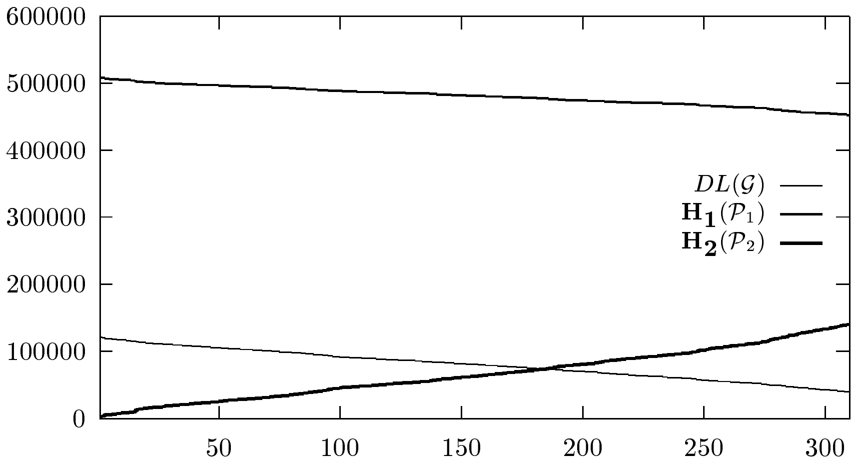

Fig. 3. Brown 10000: Description Length Terms after each merge

an easily computable heuristic approximation to ΔDL due to Solomonoff [11]. Unfortunately, we do not arrive at very good rules using this approximation (the reader who wants to see these rules nevertheless is referred to [6]). Rule constructing in its present simple form does not really work very well. It is not clear at this point whether the approximation is simply not good enough, whether a totally greedy approach does not work well or whether our training corpora do not contain enough internal structure to make 'sensible' grammar rules worthwhile. In relation to this latter conjecture it should be noted here that in experiments with *toy grammars* much better grammar rules were formed.

6 Related Approaches

Our approach is an extension to and a more exact version of Solomonoff's original one. However, there is also a clear relation to the many statistical approaches to word classification. We will give a detailed comparison to the one that seems most similar to ours:

A Generalization of a Statistical (ML) Approach

Brown *et al.* [1] give an approach to word classification based on 2-grams and the Maximum Likelihood (ML) Principle, a very common tool in statistics. In [6], we explicitly prove that our approaches reduces to Brown *et al.*'s if one 1) 'forbids' constructs and lets only merges take place (as we did in our first experiment) and 2) neglects the term of the description length of the theory $DL(\mathcal{G})$ and only looks at the description length of the data $DL(t_1^L \mid \mathcal{G})$. To give at least some idea of why this should be so: if all hypotheses one takes into

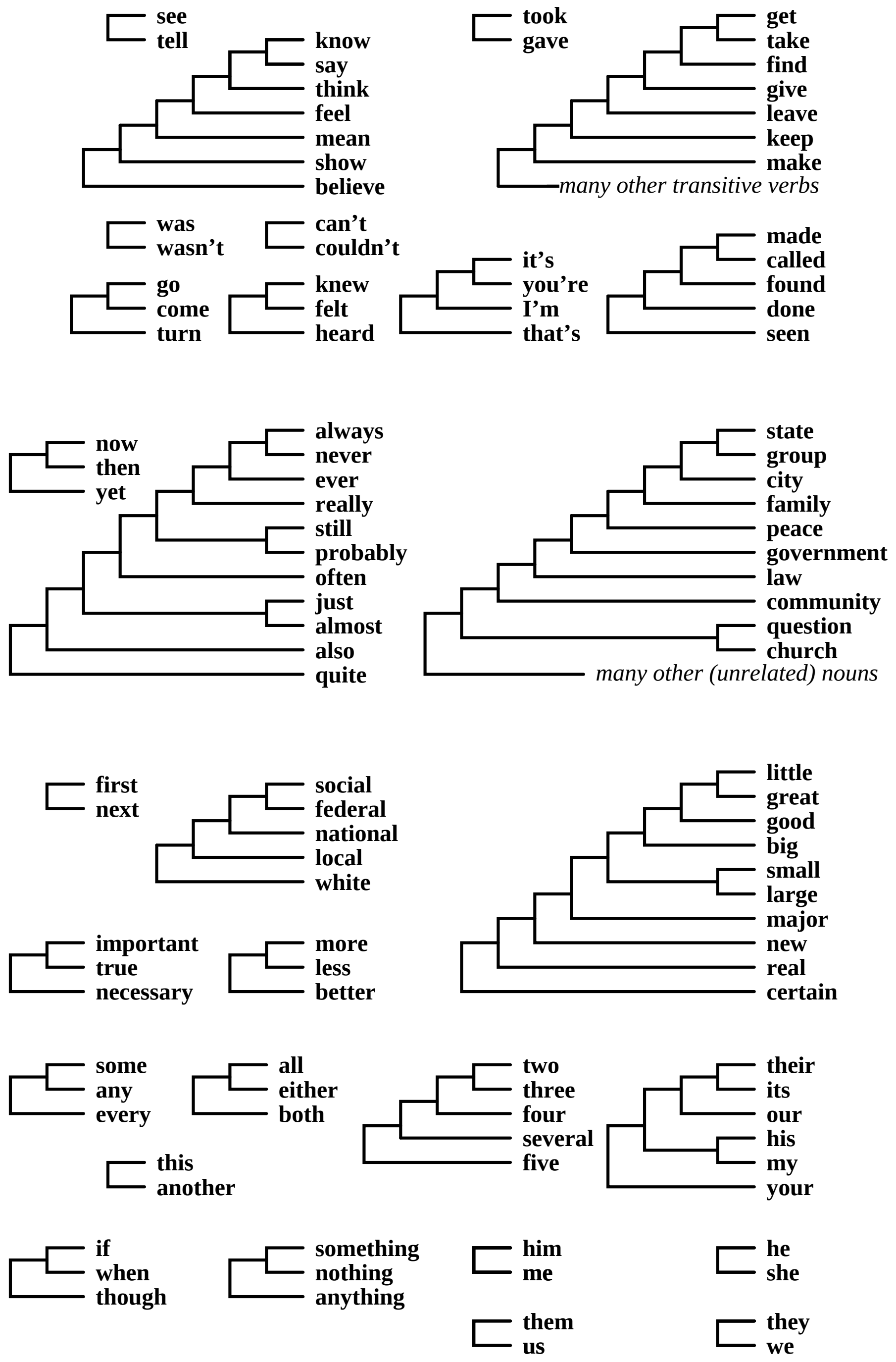

Fig. 4. Brown 10000: some of the found classes

account have the same description length, then the MDL Principle itself already degenerates into the ML Principle [10]. Neglecting the hypothesis (grammar) length is equivalent to assuming that all hypotheses have DL 0. We have already seen that a normalized variety of our parameters θ_{ij}, ν_{ij} can be interpreted as estimating the probabilities that some word or class is directly followed by some other one in a natural language text that is comparable to the training text. On these probabilities, the ML Principle can be used to find a partition of words into classes that maximizes the mutual information [5] between classes.

We have done some experiments to compare performances, and it turns out that involving the DL of the grammar yields a significantly better classification of words into classes than neglecting it (or equivalently, than using an ML approach). Another important difference between the two models is that in the ML approach, at each merge step the likelihood can only decrease. According to [1], one wants to find the partition of words into classes that gives the maximum likelihood, but it is easy to prove [6] that this is always the trivial one in which each class contains exactly one word. So in Brown's approach, it is not at all clear when one should *stop* merging classes! The availability of a clear stopping criterion seems to us an important advantage of using the MDL Principle[6].

Other Related Approaches

Finch and Chater [4] introduced another statistical approach using 3-grams, Rank Correlation Coefficients and hierarchical cluster analysis. While on the surface quite different, this approach is very similar to Brown's and ours, as we show in [6]. In this approach, too, it is not clear when to stop putting words into classes.

G. Wolff [13] gives an approach making explicit use of data-compression. This approach is very similar to ours too. It has an important extra feature called 'grammar rebuilding' which essentially involves removing words out of classes or removing production rules if this can help reduce the description length. This opens up more ways to travel through the hypothesis space. The workings of Wolff's model are much less efficient however. This seems to be one of the reasons it has not been tried on real natural language texts.

Stolcke [12] developed a Bayesian approach to Grammar Inference using an MDL prior. However, he does not take the description length of the parameter sets into account; his notion of description length is only about the *structure* of the grammar that is to be inferred. On the other hand, he uses (variants of) the *EM*-algorithm [3] to deal with ambiguous grammars, which gives an important advantage over our approach; we return to this in the conclusion.

[6] We should add here that in the implementation of our algorithm, we have made extensive use of a clever idea introduced by Brown *et al.* that makes their - and our - algorithm considerably more efficient than it would be under a straightforward implementation.

Linguistic Aspects

As we have remarked before, our approach only exploits dependencies between (non-) terminals and their immediate successors in the training text and in the parse tree. Of course, we know since Chomsky [2] that dependencies between words within a sentence may be of arbitrary distance. In a corpus-based probabilistic approach like ours it gets even worse, because the probability of a word occurring on one sentence is influenced by words occurring many sentences earlier. But notice that, at least in theory, our approach can cope with this in a peculiar manner: it starts out with only 2-gram dependencies. But as the algorithm performs more and more constructs, longer and longer parts of a sentence get treated as one single entity. Then the algorithm also considers the direct successors of this entity, thereby implicitly employing long-range dependencies. On the other hand, the question remains whether a greedy approach like ours can ever find the *right* long-range dependencies by starting out with only very short-range dependencies. It could very well be that repeated constructs, even of a more sophisticated kind than ours, will always lead $\mathcal{C}$ into a 'local minimum' at which some (construct-) grammar rules that reduce the DL have been found, but not those from which the algorithm can continue to construct more general grammar rules.

7 Conclusion and Future Work

On the practical side, we have seen that a greedy MDL-based approach to clustering words in classes works quite well, though the direct extension of this approach to the construction of simple grammar rules with more than one symbol on the right-hand side does not lead to very good results.

On the theoretical side, we have seen that several existing models both of grammar inference and of word classification can be seen as restricted versions of a more general model based on the MDL Principle, and we have introduced a clear stopping-criterion for the 'greedy algorithms' used in these models.

Apart from the problems with 'constructs', there is still another important problem with our approach, a problem that comes in two guises:

1 class per word The words of the input alphabet are *partitioned* into classes. But in natural language, a word often belongs to more than one 'class'.

Ambiguity The description length of the training text given $\mathcal{G}$ is, as it stands, only well-defined if $\mathcal{G}$ is unambiguous, i.e. (partially) parsing t_1^L according to $\mathcal{G}$ does not yield more than one shortest concatenation of classes. But many interesting context-free grammars are ambiguous at some point.

In fact, the first point made above can be seen as a special case of the second point, namely the case when we use the restriction of our algorithm that performs only merges. We are currently studying the EM-algorithm and other techniques used by Stolcke [12] in order to try to resolve this problem.

8 Acknowledgements

The author wishes to thank his supervisor Prof. Dr. Ir. P.M.B. Vitányi for many useful remarks on his Master's Thesis. Further thanks are due to Mark Steijvers and Jeroen van Maanen for many useful discussions.

References

1. P.F. Brown, V.J. Della Pietra, P.V. deSouza, J.C. Lai, and R.L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18:467–479, 1992.
2. N. Chomsky. *Syntactic Structures.* Mouton, The Hague, 1957.
3. A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 34:1–38, 1977.
4. S. Finch and N. Chater. A hybrid approach to the automatic learning of linguistic categories. *AISB Quarterly*, 78:16–24, 1991.
5. R.G. Gallager. *Information Theory and Reliable Communication.* Wiley, New York, 1968.
6. P.D. Grünwald. Automatic grammar induction using the MDL Principle. Master's thesis, Free University of Amsterdam, Amsterdam, 1994.
7. H. Kučera and W. Francis. *Computational Analysis of Present Day American English.* Brown University Press, 1967.
8. P. Langley. Machine learning and grammar induction. *Machine Learning*, 2:5–8, 1987. Editorial of special issue on language learning.
9. M. Li and P.M.B. Vitányi. *An introduction to Kolmogorov complexity and its applications.* Springer-Verlag, 1993.
10. J. Rissanen. A universal prior for integers and estimation by minimum description length. *Ann. Statist.*, 11:416–431, 1982.
11. R.J. Solomonoff. A formal theory of inductive inference, part 1 and part 2. *Inform. Contr.*, 7:1–22, 224–254, 1964.
12. A. Stolcke. *Bayesian Learning of Probabilistic Language Models.* PhD thesis, ICSI, Berkeley, 1994.
13. J.G. Wolff. Language acquisition, data compression, and generalization. *Language and Communication*, 2:57–89, 1982.