Topological Stochastic Grammars

Michael Alder
Department of Mathematics
Centre for Intelligent Information Processing Systems
The University of Western Australia
Nedlands W.A. 6009, AUSTRALIA

April 2, 1994

Abstract

King Sun Fu [1] has discussed the need for recognising that human pattern recognition is syntactic. When the human eye reads a character, there is evidence that the character is decomposed into 'strokes' and that the relationship between the components is coded in some way.

Fu attempted to tackle the problem of finding structure in objects in images by coding judiciously selected parts of the image as symbols; objects might then be described by concatenating symbols to strings. The string representing one object would, if the selection of primitive symbols was well chosen, be similar in some way to a string representing a similar object. The Fu program then required the set of strings describing a class of objects to lead to the inference of a grammar, an algorithm for deciding if a new string belongs in the 'language' of such object descriptors. Extensions to the stochastic case, where strings are assigned a frequency or probability of production by the grammar were also tried. This has been moderately successful in the form, *inter alia*, of Vector Quantisation followed by Hidden Markov models for Speech Recognition.

Defects of the general program are readily apparent: the quantisation to symbols from a finite alphabet can constitute premature decisions which prejudice subsequent classification; metric structure which is pertinent has been thrown away, and total orderings have little power.

In this paper we study a class of grammars in which an alphabet of symbols is a topological space, in our case $\mathbb{R}^n$ for some n. The concatenation operation on symbols is replaced by metric stuctures, rewrite and inference of grammars is replaced by *DownWrite* and *UpWrite* processes. The 'chunking' process which aggregates character strings into words in Natural Language text goes over to a similar entropic criterion in the continuous case. The process has been used for a variety of recognition tasks, from identifying Kanji characters by their stroke structure, to classifying moving objects in video images.

1 Introduction

The essence of a (classical) phrase structure grammar is that there are a finite set of *alphabets*, where an alphabet is a set (frequently finite) of elements called *symbols*, and where we have a map or maps from some subset or subsets of the strings of one alphabet to the symbols of another. The inverse to each map is a relation since the map is normally many-one, and any element of this relation is called a *production.* Thus, for example, the system with initial alphabet $\{S\}$, terminal alphabet $\{a, b\}$, intermediate alphabet $\{A, B\}$ and language (set of strings on terminal symbols) given by $\{aba, abba, abbba, \cdots, ab^n a\}$ would commonly be expressed by the set of productions:

$$\begin{aligned} S &\longmapsto ABA \\ A &\longmapsto a \\ B &\longmapsto b \\ B &\longmapsto bB \end{aligned}$$

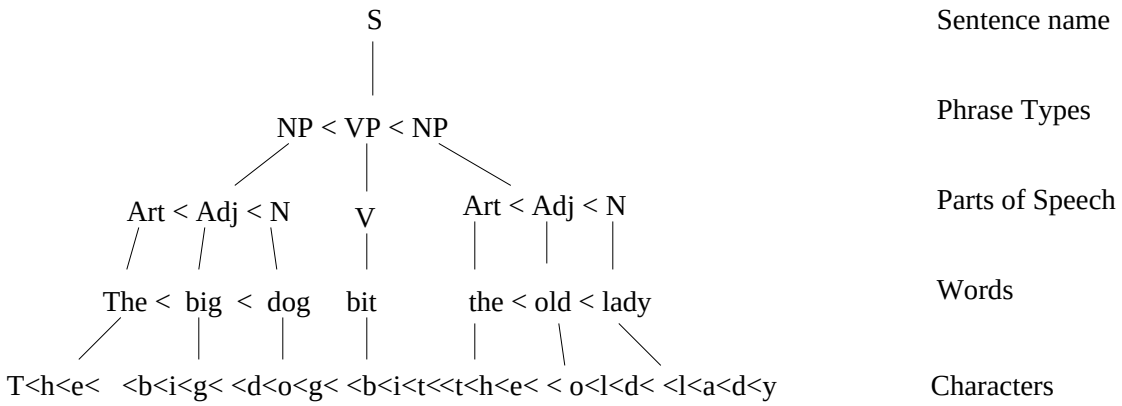

Figure 1: (Part of) a hierarchical grammar

Then we have that a single lowest level map takes the singleton string a to A and takes b^n to B, for every positive integer n. The next level map takes the string ABA to the symbol S.

The example is not entirely trivial because the levels are not clearly distinguished; in the case of the simple rewrite grammars first investigated for natural languages, there was an explicit hierarchy. S, a sentence was rewritten to, say, NP < VP < NP, (NounPhrase, VerbPhrase, NounPhrase). NP might be rewritten to Art<Adj<N (article, adjective,noun), which in turn could be rewritten to **The<big<dog**, which, with suitable rewrites for the remaining terms, produced a sentence of English. It is not unreasonable, although unconventional, to point out a further level of rewriting to characters. The order symbol < is shown explicitly because it needs to be seen that it is preserved by the mapping process.

In diagramatic form the process is sketched in *fig*.1, where it will be observed that we have five levels of the structure. At the top level we have an alphabet consisting of a single symbol, S, which we can construe as standing for *Sentence*, at the second level down we have a sequence of three symbols from a different alphabet; we shall regard this as a string from a language over an alphabet containing at least the two symbols NP (Noun Phrase) and VP (Verb phrase). Below that we have the third level with again a string from a language over yet another alphabet, at the fourth level the alphabet is the set of words of the English language, and at the fifth and bottom level, the alphabet is the set of characters from what is often, if improperly, called the Roman alphabet. Since spelling has been constrained by prescriptive grammarians, there are few options on how to rewrite a symbol at the fourth level to a string at the fifth, but it is worth, for our purposes, distinguishing between a word and its conventional representation as a string of letters.

Such grammars we shall call *hierarchical grammars*, and they are evidently about the simplest things deserving to be called 'grammars'. The formal language on the alphabet of terminal symbols $\{a, b\}$ given above can easily be made into a hierarchical grammar by deleting the last two lines and replacing them by

$$B \longmapsto bb^\star$$

where $b^\star$ denotes the set of strings consisting of iterates of b including the empty string with zero iterates. The above grammar is, of course, finite state, and it is not too hard to persuade oneself that all finite state grammars can be modified mildly to become hierarchical grammars.

In what follows we shall generalise hierarchical grammars to the case where each level alphabet is a topological space. It will simplify the exposition if we generalise only a limited class of grammars, which explains our starting point of hierarchical grammars. We consider therefore only grammars in which there is a finite set of alphabets, an order on the set so that the terminal symbols are the lowest level alphabet

and the initial symbol the highest, and where the productions are always of the form of the inverse of a map from substrings of the lower alphabet to the symbols of the next higher alphabet. This is restrictive but not intolerably so, and makes the analysis more tractable. The map from one level to the next higher level is called an *UpWrite*, and the converse relation a *DownWrite*. The problem of inferring a grammar for a language is then one of devising suitable intermediate alphabets so as to factorise the generally impractical map from the language to the initial symbol, through some intermediate alphabets and languages. In this form the problem is tractable; instead of focussing on the grammar class, we focus on the resources available to accomplish inference.

By assigning probabilities when there is a choice of rewrites, as for example by making the two productions with B on the left hand side equally probable, we generate a stochastic language. In this case, the string $\{ab^n a\}$ has probability $\frac{1}{2^n}$. In general, in a stochastic grammar, we may want at each stage to have a probability distribution over the set of strings which map to a given symbol in the higher level alphabet. This allows us to regard each map inverse as a random variable. More useful cases of stochastic grammars are the *n*gram language models for words in natural language and the Hidden Markov models for phonemes, both in use at, for example IBM's Thomas J. Watson Research Center in the Automatic Speech Recognition group, [2].

It is this framework which is the starting point for generalising to Topological Grammars. We shall allow the alphabets to be not merely finite sets but topological spaces, we shall permit the topogical structure of the spaces to replace the particular case of an order on the symbols which gives the special case of string grammars. Indeed we might contemplate the other objects such as graphs and arrays which have been also treated syntactically as other special cases.

It is found that the same mechanisms which make inference practicable for hierarchical string grammars permit inference in a general class of grammars. Abstraction for its own sake is sometimes an arid preoccupation, but abstraction which formalises analogous operations in different domains allows us to transfer insights from one to another. This is the justification for our treatment.

2 Hierarchical String Grammars

First some standard material revisited and a few somewhat less standard terms defined:

Definition 2.1 *A* Set alphabet *is any set, and a* set language on alphabet A *is any subset of the power set* $\mathcal{P}(A)$.

We follow von Neumann and define a natural number as the set of all its predecessors, and the number 0 as the empty set. We use the symbol $\mathbb{N}$ to denote the set of all such natural numbers. This allows:

Definition 2.2 *A* string *on the set alphabet A is a pair* $(n \in \mathbb{N}, f \in A^n)$*, where* A^n *denotes the set of maps from n to A.*

The set of all strings on A is widely denoted by $A^\star$. The case $n = 0$ deals with the empty string, after some minor details have been adjusted.

A *string language* over (set) alphabet A is a subset of $A^\star$. We use the notation $[a..b]$ to denote the set of integers between a and b: $\{x \in \mathbb{N} : a \leq x \leq b\}$

Definition 2.3 *A* Hierarchical String Grammar *is a finite sequence of (set) alphabets* $X_1, X_2, \cdots, X_k$*, and for each* $j \in [2..k]$ *a string language* L_j *over* X_j*, and for each string* $S_j \in L_j$ *for* $j > 1$*, a partition of the string,* $S_j = s_j^1 s_j^2 \cdots s_j^m$ *into substrings, where concatenation is used conventionally, and a map from the partition elements to the alphabet* X_{j-1}*. The image of a map from a string of* L_j *is required to be a string of* L_{j-1} *for all* $j \in [2..k]$*, the order relation being preserved.*

Definition 2.4 *The maps from the partition elements to the preceding alphabets are called the* UpWrite *maps. The relations which are the map inverses are known as* DownWrites

Remark The usual way round would be to define the DownWrites as the set of options for rewriting a symbol in a string at one level to a string at a lower level, and we normally permit mixtures of levels. It is also conventional to restrict ourselves to three alphabets, a singleton alphabet with an initial symbol in it,

a terminal alphabet, and an intermediate alphabet. Since we are much more concerned with the inference process, or the *UpWrite* in our terminology, and since the process is already quite complicated enough, we persist in our approach.

Remark We see that there are two issues which become rather more clear in the present framework than the traditional one: both are associated with the partitioning of the string in a language at any level into substrings which are then assigned a symbol in the higher level alphabet.

The first consideration is, how do we choose to partition each string into well chosen substrings? The second is, which substrings get assigned to the same symbol?

These are issues of practical import if we wish to accomplish inference. The issues are also of concern in the wider context of cognitive science where they relate to the matter known as *chunking* or to the neurophysiologists where it is known as *binding*. As we shall see, these matters and many others may be treated in the same formalism. In particular, the criteria we shall articulate have to do with extremes such as the optimal decomposition of a program into procedures and the separation of an image into parts.

3 Abstract Hierarchical Grammars

3.1 Some Category Theoretic Language

We wish to deal not merely with strings of symbols but more or less arbitrary relations between elements of a finite set. Putting them in order and allowing repetitions is going to recover the familiar world of strings, regarding them as points in a vector space will allow us to deal usefully with images. I say little here about other applications.

Let $\mathcal{S}$ denote the category of sets and set maps in the sense of [3]. Recall that an object P in a category is called a *generator* for the category iff whenever $f, g : X \rightrightarrows Y$ are different, there is a map of the category $i : P \longrightarrow X$ such that fi and gi are different. A set containing a single point is a generator for the category of sets, and all non-empty sets are generators of $\mathcal{S}$. $\mathbb{Z}$ is a generator for the category $\mathcal{G}$ of groups and homomorphisms, as is any other free group.

More weakly, a category $\mathcal{C}$ has a *set of generators* iff there is a set of objects, U, in the category such that for any pair of maps $f, g : X \rightrightarrows Y$ in the category, there is some $i : P_i \longrightarrow X, P_i \in U$ with the required property that $f \neq g \Rightarrow fi \neq gi$. A *minimal* generating set is one such that no proper subset is a generating set. By abuse of language, we may call any object in a minimal generating set a generator. It is easy to see that in the small category of strings over a set alphabet A with inclusion maps, the set of singleton strings of A (conveniently written A^1) is a minimal set of generators. It is easy to confound A and A^1, and generally harmless, so we shall do it rather casually in what follows.

The following definitions are less canonical:

Definition 3.1 *An object P in a category is called a* primitive generator *or* symbol *iff P is a generator and no subobject of P is a generator.*

Definition 3.2 *A set of objects U in a category is a* set of primitives *or a* set of symbols *iff first it is a minimal generating set for the category, and second, no object in the set has a subobject such that replacing the object in the set by the subobject also gives a generating set.*

It is easy to see that any singleton set in the category of sets, $\mathcal{S}$, is a symbol, that $\mathbb{Z}$ is a symbol in the category $\mathcal{G}$, and that the alphabet is a set of symbols in the category of strings over that alphabet and inclusions thereof. In the category of strings and inclusions over some alphabet, the set of bigrams of the language, all substrings of length 2, are also a set of generators, but not a set of symbols.

In the category of finite subsets of $\mathbb{R}^2$ and their isometries, each point of $\mathbb{R}^2$ is a symbol. We shall allow the word 'symbol' to also be used of any element of a set of symbols in a category.

The following is somewhat informal but will suffice for present purposes:

Definition 3.3 *A category $\mathcal{C}$ will be called a* category of sets with structure *whenever there is a (Forgetful) functor from $\mathcal{C}$ into $\mathcal{S}$.*

The familiar categories of groups and homomorphisms, $\mathcal{G}$, and of topological spaces and continuous maps, $\mathcal{T}op$, are examples of categories of sets with structure, as is, rather trivially, $\mathcal{S}$. We shall be particularly concerned with the small categories of finite subsets of a finite dimensional linear space and the isometries between them, $\mathcal{L}in_n : n \in \mathbb{Z}^+$, which also have this property. It is natural to think of the category of finite subsets of $\mathbb{R}^2$ as a set of binary images, by taking the set of black pixels on a white ground (or the converse) and specifying the set of pixel locations and regarding the (integer) coordinates as points of $\mathbb{R}^2$. We shall regard this as a hint at some motivation for the treatment.

Definition 3.4 *If $\mathcal{C}$ is a small category of sets with structure, a* $\mathcal{C}$-language *is a subcategory of $\mathcal{C}$. Generally, an* abstract language *is a $\mathcal{C}$-language for some small category $\mathcal{C}$ of sets with structure.*

Definition 3.5 *A $\mathcal{C}$* term *of a $\mathcal{C}$-language $\mathcal{L}$ is an object in a $\mathcal{C}$-language. A* subterm *is a subobject.*

Definition 3.6 *The $\mathcal{C}$* alphabet *for a small category $\mathcal{C}$ of sets with structure is the set of all symbols of the category.*

Hence we may talk about the alphabet for a language in a much more general sense than is usual. It is easy to see that there always is such a set. In the case of the string languages of strings of symbols from an alphabet, we have recovered the usual terminology of formal language theory. An alphabet is an alphabet, a language is a language. But we have also found a geometric exemplar of the same abstract structure: A set of binary images is also a language with the points of $\mathbb{R}^2$ defining the alphabet. There are many other examples, and the formal similarities may be taken very much further and exploited.

Recall that a *coproduct* of X and Y , objects in any category $\mathcal{C}$, is an object $X \coprod Y$ and maps $i_X : X \longrightarrow X \coprod Y, i_y : Y \longrightarrow X \coprod Y$, such that if W is any other object of the category and $\alpha : X \longrightarrow W, \beta : Y \longrightarrow W$ are any maps, there exists a unique map $j : X \coprod Y \longrightarrow W$ such that $ji_X = \alpha, ji_Y = \beta$.

In $\mathcal{S}$, the union of two sets is the coproduct, it $\mathcal{G}$ it is the free product. We weaken the notion of a coproduct to a *concatenation*:

Definition 3.7 *If X and Y are any two objects in a category, a* concatenation *of the pair is an object $X \bigwedge Y$ and maps $i_X : X \longrightarrow X \bigwedge Y, i_y : Y \longrightarrow X \bigwedge Y$ such that if W is any other object of the category and $\alpha : X \longrightarrow W, \beta : Y \longrightarrow W$ are any maps, then there is at most one map $j : X \bigwedge Y \longrightarrow W$ such that $ji_X = \alpha, ji_Y = \beta$. In other words, j may not exist but if it does it is unique.*

Clearly any coproduct is a concatenation, and the concatenation of two strings in the ordinary sense is a concatenation in the above abstract sense in the category of strings over some alphabet and their inclusions. The converse is not the case: if the last symbol of string X is the first symbol of string Y, the maps α, β may overlap into a concatenation in the abstract sense, but not of course in the conventional sense.

Plainly, a concatenation of two objects is not generally unique when it exists. It is easy to see how to define concatenations of more than two objects.

Definition 3.8 *A $\mathcal{C}$-language $\mathcal{L}$ admits a* word structure *$\mathcal{W}$ when there is a set, $\mathcal{W}$, of subterms of the terms of $\mathcal{L}$ such that each term of $\mathcal{L}$ is a concatenation of elements of $\mathcal{W}$. An element of $\mathcal{W}$ is called a* word.

All languages admit word structures in rather trivial ways, but one has the instinct that some word structures are better than others, and that there are some languages with rather good word structures. For example, in English text at the letter level, there are substrings of the sentences called, surprise, surprise, words, which comprise rather satisfactory substrings. Looking at some images, say images of characters from some font, Fu had the instinct that there are word structures here also. We are observing that in this case, we are asking for decompositions into subsets.

Definition 3.9 *An* UpWrite *between a $\mathcal{C}$-language $\mathcal{L}$ and a $\mathcal{D}$-language $\mathcal{M}$, is a word structure $\mathcal{W}$ on $\mathcal{L}$, and a map from $\mathcal{W}$ to the alphabet of $\mathcal{M}$ which yields a functor from $\mathcal{L}$ to $\mathcal{M}$.*

Definition 3.10 *The* DownWrite *corresponding to an UpWrite is the relation between the alphabet of $\mathcal{M}$ and the set of all subterms of terms of $\mathcal{L}$ which is inverse to the map constituting the UpWrite.*

Definition 3.11 *An* abstract hierarchical grammar *is a finite sequence of abstract languages* $\mathcal{L}_j : j \in [1..k]$ *together with an UpWrite from* $\mathcal{L}_j$ *to* $\mathcal{L}_{j-1}$ *for all* $j \in [2..k]$.

To mitigate the anguish of abstract definitions, some examples:

Example: Let the top level language have alphabet the words of the English language and strings the allowable sentences. let the lower level language have alphabet the characters and punctuation that we call the Roman alphabet. The word structure is the decomposition of the substrings between whitespaces and other punctuation, the UpWrite assigns each such string to a word. Because of the depredations of prescriptive grammarians, the map which assigns strings to words is mostly one to one, although there are occasional words with several admissible spellings. Due, perhaps, to the depredations of the illiterate. The UpWrite in this case merely assigns the word to its spelling and the DownWrite the spelling to each word.

Example: Let $\mathcal{L}_1$ denote the set of strings of length 1 drawn from the alphabet comprising the single symbol S. There is one object in the category, and one map, the identity map. It is the second most boring category there is. Let $\mathcal{C}_2$ be the category of all strings of symbols on the alphabet A,B together with the inclusion maps. Let $\mathcal{L}_2$ be the subcategory consisting of the string ABA and the identity map. The UpWrite from $\mathcal{L}_2$ to $\mathcal{L}_1$ is the map sending (all) the single string ABA to the symbol S. The associated functor sends the string ABA to the string S of length one. Let $\mathcal{C}_3$ be the category of all strings on the alphabet a,b, and let $\mathcal{L}_3$ be the subcategory comprising the strings $\{ab^n a : n \in \mathbb{Z}^+\}$, and let us choose subterms to be the singleton string a, and the strings b^n for all positive integers n. Let the UpWrite map send a to A, and b^n to B, so the functor sends, for example, *abbbba* to ABA.

Example: Let Let $\mathcal{C}_1$ be the set of all finite subsets of the vector space $\mathbb{R}^6$ and the isometries between them. Let $\mathcal{L}_1$ denote the subcategory of subsets of $\mathbb{R}^6$ containing one element, and observe again that this set is a set of generators for $\mathcal{C}_1$ and hence the set constitutes an alphabet for the full language $\mathcal{C}_1$ (and others).

Let $\mathcal{C}_2$ be the set of all bounded subsets of $\mathbb{R}^2$ and the isometries between them, which we shall regard as containing and completing the set of all possible binary images, as mentioned earlier. Let $\mathcal{L}_2$ be some particular collection of figures I shall call *lines* and *disks*. I take the set of pairs of points $\{(\mathbf{p}, \mathbf{q}) \in \mathbb{R}^2\}$ and the line $\mathbf{pq}$ is the set of points on the line segment between them. Similarly, I take a pair consisting of a point $\mathbf{p} \in \mathbb{R}^2$ and a positive real number r; then the disk $(\mathbf{p}, r)$ is the set of points of distance less than r from $\mathbf{p}$.

The language $\mathcal{L}_2$ has terms consisting of all such lines and disks, together with isometries between them. The points of $\mathbb{R}^2$ comprise the alphabet (or atomic subterms, as it may be convenient to think of them).

An UpWrite from $\mathcal{L}_2$ to $\mathcal{L}_1$ may be constructed by taking each term T, either a line segment or a disk, and computing the six real numbers:

$$\int_{\mathbb{R}^2} \chi_T \begin{bmatrix} x \\ y \end{bmatrix} x^n y^m \, dx \, dy$$

where χ_T is the characteristic function for the set T and $n, m \in \mathbb{N}, n + m \leq 2$. That is, the zeroth, first and second order moments for the term T. Central moments might have been a better choice, of course, but the principle is clear. Each set is taken into an element of the alphabet of the higher level space. The UpWrite map and the associated functor are indistinguishable in this case because the decomposition to subterms never decomposes a term at all. In this case also, the word structure lacks a certain interest.

It would be easy in principle to have a different arrangement whereby the terms of the language $\mathcal{L}_2$ would comprise the disks, the line segments, and the disjoint unions of such elements, whereupon the same scheme would give either singletons or pairs of points in the language $\mathcal{L}_1$. The word structure would be somewhat less trivial, but still hardly exciting. Fortunately there are more interesting possibilities.

Remark Computing a vector to describe a set of points in $\mathbb{R}^2$ may be done in various ways; many of them reduce to projecting the characteristic function of the set onto a suitable basis of functions, as in the last example.

Remark A hierarchical grammar consisting of just two levels in the hierarchy, the base level consisting of the objects and the top level consisting of a coding of the objects into symbols or constellations of symbols, leaves a bit to be desired in excitement and novelty. We are going to be more interested in a process which naturally admits more levels than this in general. I shall next show how to accomplish this in special cases, transferring some ideas from the case of string grammars to linear grammars. Finally, for a somewhat exotic example:

Example: Let the top level be a small category of Lie groups and Lie group homomorphisms; each such group decomposes as a set of matrices with an algebraic structure of group multiplication. Each matrix is an array of complex numbers, and each complex number is a pair of real numbers together with a rule for multiplying them. One could go further with this decomposition were one so inclined. The base language has alphabet perhaps the ten digits and a decimal point, an enormous set of infinite strings of symbols from this alphabet arranged in some very peculiar ways, and is organised into intelligibility only via the grammar.

3.2 Topological Grammars

The case which is to be considered here is where the alphabets are all topological spaces, and we shall specialise this even further in applications.

Definition 3.12 *A* hierarchical topological grammar *is a hierarchical grammar in which each alphabet is a topological space, each language is also a topological space, and each UpWrite map is continuous.*

For the UpWrite map to be continuous, we need a topology on those subsets of the domain topological space which are subterms of the domain language. This is easily ensured when the subsets are finite.

Definition 3.13 *A* topological stochastic grammar *is a topological grammar in which each term may be assigned a probability, or more generally, there is a probability measure on the space of terms for each language, and also on the words in the word structure at each level; the UpWrite has to preserve the probability in an obvious sense.*

Remark: Just as in the case of strings, there is a necessary consistency condition for the general case in an abstract category; if A is a subterm of B, then A must occur at least as often as B and must have at least equal probability. Similarly, just as a local predictor, conditioned on some part of the string and estimating the probability of the rest, can be built up from a probability for each substring and in turn leads to such a probability distribution, we have that more generally in the category setting we can distinguish between predictors and stochastic grammars but that they are equivalent, in that given one we can obtain the other. Given two consecutive levels of a topological grammar, there are two ways of computing the probability of a string of words: one is to find the probability of the string at the word level, one is to find it at the character level. These have to give the same answer in any sane system.

A little thought suffices to convince oneself that this includes as rather trivial special cases the stochastic grammars mentioned above, the $n$grammars and the Hidden Markov models.

Definition 3.14 *A* grammatical inference engine *is an algorithm which when presented with a language on some alphabet, produces a hierarchical grammar with a singleton alphabet at the top level, and which produces the given language by the sequence of DownWrites.*

The trivial grammar which does it all in one step is always a solution, but has little importance. The interesting grammars which are sought in practice allow one to generate the next stage down by simply computable functions. As an example, suppose we are given a lowest level consisting of input-output pairs of symbol strings on the finite alphabet of ASCII characters. We may regard this as a functional specification of a program by supposing that we are required to obtain a program which given an input will produce the required output. We may stipulate that the next level up is required to contain only the primitive functions of PASCAL. Then the task of finding a grammar for the desired input-output set is known as programming, and the top level singleton will be called a program.

This should suffice to establish that the programmer may be called upon to become a grammatical inference engine, and that we do not know much about such things in general. As another example, perhaps whimsical, consider the process of learning to drive a car. For the beginner, learning to turn left requires remembering to switch on the traffic indicator, check in the driving mirror, put ones foot on the clutch and change gear, turn the steering wheel, brake and check for pedestrians. The act of depressing the clutch requires learning too, as a sequence of muscle contractions which the neonate has to acquire. After sufficient practice, turning left becomes an automatic act and driving a long distance is a sequence of similar acts. The singleton top level of driving from home to the office has been factorised into a sequence of choices

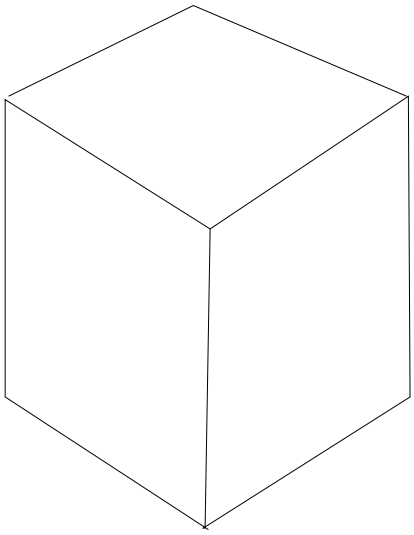

Figure 2: A binary image

of a particular DownWrite which has muscle contractions as its lowest level. The phenomenon known to psychologists as chunking is the process of constructing a suitable grammar.

Picture grammars and graph grammars are examples which demonstrate that the restriction to string grammars is impractical, and that some degree of generalisation is warranted.

4 Linear Grammars

4.1 Introduction

The generality of the last section may be perceived as excessive, and is provided only to show that the ideas are coherent, and to permit the transfer of ideas and methods between domains. We specialise now to place additional restrictions on topological grammars in order to show that the structures discussed are not vacuous.

Definition 4.1 *A* linear grammar *is a topological grammar in which each alphabet is* $\mathbb{R}^n$ *for some non-negative* n*.*

An example of a grammar of some practical interest arises in image understanding. Suppose we have a binary image such as *fig.2.* We shall consider this to be a finite subset of $\mathbb{R}^2$.

To the eye of the average human being, this is clearly a picture of a cube. The eye has a top level object consisting of an entity called a cube, the cube is decomposed, perhaps, into faces which are roughly parallelograms, each parallelogram decomposes into edges, some being shared, and the edges or line segments decompose into pixels, the primitives or terminal symbols. This decomposition is, of course, by no means unique, but it clearly constitutes a grammar in our sense.

It is easy to choose to measure or describe a pixel as a point in $\mathbb{R}^2$, and a line segment as a point in $\mathbb{R}^4$, by giving its length, centre and angle of orientation with respect to the X axis, for example, and if we suitably code parallelograms as points in $\mathbb{R}^n$ for suitable n, and a cube likewise as a point in some other space $\mathbb{R}^m$, then the decomposition has the form of a linear grammar. It is not difficult to devise suitable coding schemes for the higher level objects. Moreover, if a slightly different image, also of a cube, were given, an inference engine might be constructed so that the final points denoting the cube are close in the top level space. Such an inference engine would be said to recognise an image as containing a cube, if the final UpWritten points are sufficiently close to other cubes.

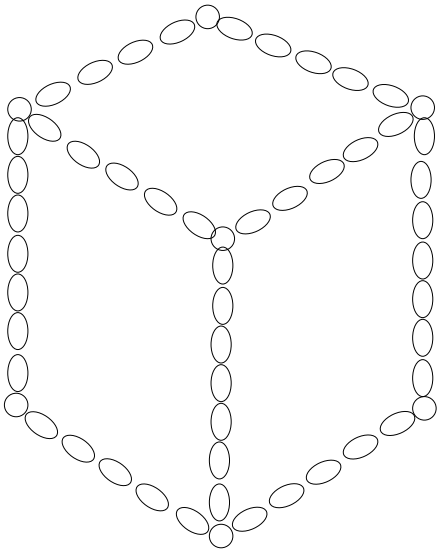

Figure 3: The first UpWrite

Any inference engine of sufficient power to accomplish this task would be a cube recogniser, and it would be of some limited interest. If it were able to be adapted to be able to perform the same process on some other class of images, it might be said to be accomplishing image understanding for this class. This would make it considerably more interesting, as it would be simulating little understood cognitive processes currently associated only with the central nervous systems of animals. We shall describe the functioning of a linear grammar which recognises cubes and which admits of such generalisation.

4.2 Inference

We represent the pixels as points in $\mathbb{R}^2$. This enables us to dispense with resolution issues at the lowest level and to show the generality of the procedure.

The first step is to select a resolution radius. This is trivial in the case of an object made out of line segments, and we shall defer the general issue. We then find a point of the set, at random if necessary, and find the subset of points of the image which lie within the ball in the space given by the resolution radius. We seek a descriptor of the resulting point set which can be expressed as a vector of real numbers and which allows allows us to generate expectations concerning other points. It is not essential that we use a parametric probability density function, and such a choice would constitute an unnecessary commitment to a model class, but modelling the data (consisting in this case of a short line segment usually) by a multivariate gaussian distribution is entirely feasible. If we conceive of the set as specified by its characteristic function, then we seek a continuous approximation to it and this is conveniently done by expressing it in a basis of functions defined on the base space or some compact region of it containing the data. We shall describe this, without prejudice, as computing moments of finite order for the data.

Suppose then that we elect to use zero, first and second order moments in order to describe the set. This amounts to counting the data points, computing the centre and also the covariance matrix for the points within a resolution radius ball. We obtain in the case of the base space in dimension two, a number, the point count, a centre given by two reals, and a symmetric two by two matrix. For reasons which we shall also defer discussing, it is convenient to represent the matrix by the two eigenvalues and the angle between the principal axis and the X-axis. This gives a total of six numbers, so we can describe our point set as a point in $\mathbb{R}^6$. This can then be repeated by subtracting the points already so described from the image, and iterating until no further points are left. Thus we have replaced the original point set in $\mathbb{R}^2$ by a somewhat smaller such set in $\mathbb{R}^6$. This constitutes the first UpWrite.

We regard the quadratic form given by the first and second moments as defining an ellipse, $\{\mathbf{x} \in \mathbb{R}^2 :$

$(\mathbf{x}-\mathbf{m})^T\mathbf{C}^{-1}(\mathbf{x}-\mathbf{m}) = 1\}$. This allows us to contemplate the result graphically as in $fig.3$, where we observe that if there are sufficient points in the original image and if the resolution radius is sufficiently large, we will have obtained some degree of data compression without doing injustice to the original data. Following Rissanen [4], it is useful to see the grammar construction as constituting a sequential data compressor. If the resolution radius is too large, the number of chunks (points in the UpWrite space) will be rather small, possibly only one, and the order of the representation might have to be increased in order to capture information about the data. This of course is how a certain class of OCR systems work: in the limit for large resolution radius, we reduce to a standard moments calculating routine. At the other end, if the resolution radius is less than the distance between pixels, the UpWrite merely duplicates the lower level representation. This corresponds to the two extreme cases of finding a word structure; the words may be the whole strings, which doesn't accomplish much, or they may be just the symbols themselves which accomplishes precisely nothing. Somewhere between the extremes may be found some place to stand which gives an improvement.

Given that we now have a set of local models which, having compressed the data, allow us to form some sort of prediction of what points of the original data set may be expected to be close. This amounts to using the ellipses of $fig.2$ as predictors of where points in the lower level space may be expected: regarding the fitting of quadratic forms as treating the modelling of the image as a gaussian mixture modelling problem [5,6], encourages this perspective, but it is not necessary. If the ellipses are seen as describing lower level structure, we can see them as predicting that a close ellipse will have a fairly well defined centre at a Mahalanobis distance of about two, and a similar orientation. If we suppose any plausible *pdf* for the location, size and orientation of a close ellipse, then we can compute the amount of information supplied by any particular ellipse relative to a fixed one (or more). This will be small if the adjacent ellipse is of about the same shape and size and orientation and has a centre of about Mahalanobis distance 2. If the adjacent ellipse is very different in any of these respects, the information supplied will be high. We do not need to compute this, which would in any case presuppose a known *pdf* for the location of adjacent ellipses, it is sufficient to set some threshold. In practice it is simpler to test to see if any significant change of orientation or location or shape occurs, where we pass over the issues of selecting a suitable threshold. We may thus agglutinate the predictors when each of them predicts the points of the other, or when the data points belonging to them are mutually predicted. In visual terms, one paints an ellipse purple, and then checks in a neighbourhood to colour an adjacent ellipse purple if the second is credibly predicted by the first, and one rejects the agglutination as unreasonable when the information supplied to the predictor by the data is over some threshold. This is, of course, just how one may extract words from text; one runs an $n$gram predictor down the text and at locations of high entropy, one segments. We have been told that this was pointed out by Shannon, but know of no reference to it.

In practice then, we colour an ellipse, say purple, and test adjacent ellipses to see if they are within the new 'resolution radius', which is obtained in principle by measuring the relations between ellipses and deriving a conditional probability distribution for changes in ellipse parameters. If an adjacent ellipse is within this tolerance, it is also coloured purple, otherwise it is assigned a different colour. One spreads out from an initial ellipse until one is stopped or there are no uncoloured ellipses. All ellipses in the UpWrite space thus acquire a colour. Finally, we seek a descriptor of each set of ellipses having the same colour. Again, we may simply compute low order moments of the point set in $\mathbb{R}^6$.

In the case of our cube, it is easy to see that we shall obtain the line segments specified as points in the third UpWrite space.

It is also easy to see that the same process chunks text at the letter level into words or stems and inflections. This is not an analogy but the same operation in a different category.

The step from level two to level three as described may entail some rather large dimension for the level three space; this may be reduced by various methods involving contraints on the moment representation. I shall omit discussion of the more practical issues in order to focus on the theoretical considerations.

The step from level two to level three is then iterated exactly, with the only issues arising being the selection of a suitable choice of resolution radius and order of representation. The task of automating these choices will be discussed elsewhere, but it is clearly data driven.

It is easy to see that a suitable choice can be enforced which will yield line quartets which constitute a parallelogram shaped face at the fourth level. Finally, UpWriting three faces of a cube in precisely the same manner yields a single point at the top level.

4.3 Recognition

The inference engine described in the last subsection has been constructed as a C program using X-Windows graphics on a SUN SPARC2 workstation. The different levels of Upwrite use essentially the same code, and differ principally in the dimension. It was 'trained' on a set of hand drawn 'cubes'. The resulting points in the top level space were found to be clustered in a lower dimensional subspace. In order to decide if a test object was a cube, it was UpWritten in the same way, and tested to see if it belonged to the cluster using a single multivariate gaussian. Results tend to agree with human perceptions of what constitute cubes, with differences explicable in terms of poor choices of order of representation (kept low for reasons of computational convenience) and choices of threshold or resolution radius, made by picking a plausible number for the same reasons. Other shapes such as tetrahedra or pyramids may be presented, and the system can then find the closest cluster of points at the top level space, thus accomplishing pattern recognition of the conventional sort.

Variants of the program have be trained to recognise Japanese characters, a version which recognises images of aeroplanes has also been successful, and Miss Sok Gek Lim has applied the same methods to images of moving objects in a domestic environment.

5 Conclusions

Generalising the process from cubes to similar geometric objects seen under projection presents no essential difficulty. Generalising to higher order structures such as objects built up from cuboids and pyramids puts severe strains on our computing resources but entails no conceptual problems. Generalisations to images constructed out of curves have been accomplished, again without conceptual strain but entailing some heavy computational demands. Invariance under translation is readily learned, as may be confirmed by examining the UpWrites of line segments, by finding that the points lie on lower dimensional manifolds; the reason for choosing to diagonalise our forms rather than, say, use matrix entries, is that submanifolds of interest are more likely to be linear.

The problem of occlusions of parts of one object by another, or of deciding to which object among several that a given subobject belongs, involve other ideas which will be reported elsewhere. Some forms of recognition in this case entail conflicts between different levels of UpWrite, a delicate matter requiring further investigation, suggesting as it does that the class of hierarchical grammars may be too circumscribed for a comprehensive account of syntactic recognition.

The inference engine described has, however, sufficient generality to be of interest as modelling concretely some aspects of cognitive processes. It was derived by contemplation of a neural model, which connection will be elaborated elsewhere, and goes some way to validating the model as a description of the central nervous system.

The program which constitutes the inference engine has been written by Mr. Robert McLaughlin, and a more general version by Dr. Chris deSilva. I am most grateful to both; I also wish to thank Professor Yianni Attikiouzel for providing the stimulating environment, CIIPS, within which the work has been carried out.

6 References

[1] Fu, K.S, *Syntactic Pattern Recognition and Applications*, Prentice-Hall, New Jersey, 1982.

[2] Jellinek, F, *Self-Organised Language Modelling for Speech Recognition* IBM Research Report, Continuous Speech Recognition Group, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y.10598.

[3] Freyd, P, *Abelian Categories* Harper and Row, 1964.

[4] Rissanen, J, *Stochastic Complexity in Statistical Enquiry*, World Scientific, 1989.

[5] Everitt and Hand, *Finite Mixture Distributions*, Chapman and Hall, 1991.

[6] Titterington, Smith and Markov, *Statistical Analysis of Finite Mixture Distributions*, John Wiley and Sons, 1985.