Mach Learn (2007) 66:93–110
DOI 10.1007/s10994-006-9592-9

Learning deterministic context free grammars: The Omphalos competition

Alexander Clark

Received: 15 June 2005 / Revised: 13 April 2006 / Accepted: 22 June 2006 / Published online: 4 August 2006

Abstract This paper describes the winning entry to the Omphalos context free grammar learning competition. We describe a context-free grammatical inference algorithm operating on positive data only, which integrates an information theoretic constituent likelihood measure together with more traditional heuristics based on substitutability and frequency. The competition is discussed from the perspective of a competitor. We discuss a class of deterministic grammars, the Non-terminally Separated (NTS) grammars, that have a property relied on by our algorithm, and consider the possibilities of extending the algorithm to larger classes of languages.

Keywords Grammatical inference · Context Free Languages · NTS languages

1. Introduction

Context Free Languages are widely used in a number of different areas, including programming languages, speech recognition, natural language processing and bioinformatics. One of the attractions of these languages is that they arise convergently from several different models: from context free grammars, considered as tree stuctures or as rewriting systems, and from push-down automata. Some interesting closure properties and the existence of efficient algorithms for parsing and generating, have also contributed to their wide diffusion. A subclass that has received special attention is that of deterministic context free languages, which have the property that they can be recognised by a deterministic push down automaton.

The learning of context free grammars has for some time been considered as a challenging problem in Machine Learning. While it seems clear that under any plausible learning criterion the entire class of context free grammars cannot be learned, there are hints that some substantial subclasses of deterministic languages can still be learnt. In this paper we

Editor: Georgios Paliouras and Yasubumi Sakakibara

A. Clark (✉)
Department of Computer Science, Royal Holloway University of London, Egham, Surrey, TW20 0EX
e-mail: alexc@cs.rhul.ac.uk

discuss an approach to this problem, that we used in our winning entry for the Omphalos competition (Starkie, Coste & van Zaanen, 2004). This work grew out of previous research that was concerned with the learning of context free grammars for natural language from naturally occurring corpora (Clark, 2001), and research into formal learnability of deterministic regular languages (Clark & Thollard, 2004a, b).

We experimented with a variety of different approaches to solving the competition problems. In describing them here, we have tried to strike a balance between presenting an accurate description of the algorithms we actually used in winning the competition, and presenting a coherent explanation of the underlying ideas. Competitions with fixed deadlines inevitably only permit a limited amount of coding, debugging and testing. In particular we do not present any further empirical work, other than that on the competition test sets. Thus many of the particular claims we make about the algorithms and the reasons for their success will not be strictly justified by the experimental results we present in this paper. Nevertheless, given the public nature of the competition, and the intention to continue with further competitions, we feel it is appropriate to give a timely exposition of our methods, with as full an explanation and justification as possible.

This paper is structured as follows: we will first discuss the competition in general (Section 2) and a few specific aspects of the problems that allow learnability (Section 2.1). We will then discuss the basic ideas behind our approach (Section 3) followed by the implementation and discussion of results (Sections 4 and 5).

2. The Omphalos competition

The Omphalos competition is described fully in Starkie, Coste and van Zaanen (2004), and here we shall merely offer a brief description, highlighting some of the key features of the competition and explaining how they guided the design and implementation of the winning algorithm.

We assume some familiarity with formal language theory (Harrison, 1978). We start by defining some common notation. We will denote a finite alphabet by Σ, the free monoid on Σ by Σ^* and the identity (empty string) by ϵ. A context free grammar G is a tuple (Σ, N, P, R) where N is a finite non-empty set of non-terminals. We will write $V = N \cup \Sigma$. We will use lower case Greek letters for elements of V^*. $P \subseteq N \times V^*$ is a set of productions which we will write in the form $X \to \alpha$, for $X \in N$, and $\alpha \in V^*$. $R \subseteq N$ is a set of sentence symbols. We say that $\alpha X \beta \Rightarrow \alpha\gamma\beta$ if $X \to \gamma$ is an element of P. We extend the single step derivation $\Rightarrow$ to a multiple step derivation $\stackrel{*}{\Rightarrow}$ recursively.

For $X \in N$, define the yield of X, $y(X) = \{\alpha \in \Sigma^* | X \stackrel{*}{\Rightarrow} \alpha\}$. The language of G written $L(G)$ is defined as $\{w \in \Sigma^* | \exists S \in R : S \stackrel{*}{\Rightarrow} w\} = \bigcup_{S \in R} y(S)$. We shall also use the term sentential form to refer to an element $\alpha \in V^*$ such that $\exists S \in R : S \stackrel{*}{\Rightarrow} \alpha$.

This slightly generalises the normal definition of a context free grammar which is defined to have a single sentence symbol: $|R| = 1$. Of course, this does not change the power of the formalism, but it is convenient for two reasons: first, we will use these grammars to represent intermediate states of the computation, where we may not yet have a sentence symbol defined, and secondly, when we consider a restricted class of grammars below, the NTS grammars, this modification does increase the class of languages allowable.

The competition consisted of a set of problems of the form (Σ, X, Y) where Σ is a finite alphabet, $X \subset \Sigma^* \times \{0, 1\}$ is a labelled set of labelled strings, and $Y \subset \Sigma^*$ is a set of unlabelled test strings. The competitors were required to label the test strings and submit them to an online oracle. The competition organisers first generated a set of target context free

Fig. 1 Summary of problems

Problem	Data	DCFL	$\|\Sigma\|$	Sample size	Solved	$\|N\|$
1	+/−	Yes	5	801	Small	7
2	+	Yes	5	280	Both	10
3	+/−	Yes	24	924	Small	13
4	+	Yes	24	418	Both	15
5	+/−	No	24	1238	Both	20
6	+	No	24	498	Small	19

grammars by a combination of random and non-random means. They then generated a set of positive strings from the target grammar, and then a second set of strings at random for some of the data sets. These examples were then labelled as positive or negative as appropriate. They then used a similar method to generate a set of test strings. The original test sets were too small and made the problem too easy, so they subsequently produced a larger set of test data which included data generated from regular approximations to the target context free grammars. This produced a set of strings that appeared similar to strings in the language thus making the labelling problem much more challenging. The criterion used for success was the exact labelling of *all* of the test strings as either in or not in the target language, which could be verified by submission to a web-based form. No feedback was provided about the accuracy of the labelling other than determining whether the labelling was completely correct or not, but multiple submissions were allowed: up to 10 per day. Only limited information was provided about the methods used to generate the grammars and the samples. Figure 1 summarises the data sets we attempted. Some much more difficult problems were also added later in the competition but we did not attempt to solve them.

A large number of different researchers downloaded the datasets but only a few submitted solutions to the oracle. The competition was in general perceived to be too difficult by the grammatical inference community. A variety of different techniques were employed as described in Starkie, Coste and van Zaanen (2004). Some of the small data sets were solved using a simple n-gram based classifier. Simple search algorithms were also employed but were only capable of solving the smallest problem.

2.1. Formal analysis

First, before committing the time and effort required for a competition of this sort it is worth deciding whether the underlying problem is inherently intractable. There are a number of aspects of the competition problems that are important. First, as a result of our personal research goals, the approaches we considered rely only on the set of positive strings. We decided to approach the problem as a classical grammatical inference problem: that is to say, we attempted to infer an explicit context free grammar, and label the test data according to whether the string was in the language defined by the grammar.

When the strings are generated randomly based on the target grammar, the distribution then gives some help to the learning algorithm. Thinking of this more formally, this means we do not need to look for a distribution free learning algorithm which would not exist for a sufficiently large class of languages (Kearns & Valiant, 1994). As was shown in Clark and Thollard (2004b), this restriction on the set of distributions is sufficient to allow learnability for the class of regular languages when the distributions also are μ-distinguishable (Ron, Singer & Tishby, 1995). In the Omphalos competition, the positive samples were generated largely from a distribution that depended on the grammars, i.e. from a Probabilistic Context

Free Grammar or approximation thereof. There were, however, a number of deviations from this strict procedure. Secondly, not only do we not have to learn with respect to every distribution, but we also do not even have to learn *all* CFGs or DCFGs. The target grammars were generated according to a largely random process. Thus to be successful we merely have to be able to learn a set of grammars that has reasonably large probability with respect to this prior probability. Even if there were only a 50% chance of a particular random grammar falling within a putative class of learnable grammars, then given that there are six small grammars in the Omphalos competition, we would still have a very good chance of success. Note that this point is completely unrelated to the "probably" part of PAC-learning, which concerns the the probability of *samples*, rather than the prior probability of *concepts*.

Thirdly, we are not restricted to a single guess. Since multiple submissions were allowed, if we have an algorithm that can produce a reasonably sized set of hypotheses then this again could be successful.

In spite of these overall positive facets to the competition, there is one overwhelming negative point. The training data is strictly limited to a very small amount. Figure 1 shows the size of the data sets compared to the problem complexities. We can see that the sample size is roughly equal to the product of the number of non-terminals and alphabet size. We were very surprised that our approach was as succesful as it was, given the extremely limited amount of data. This problem was exacerbated by the requirement for exact identification of the labelling of the large test set. In several cases there was simply not enough information in the training set to distinguish between a number of different possible grammars. We will give some examples below.

Traditional definitions of sample complexity need some modification to cope with one of the peculiarities of grammatical inference, namely the variable length of the samples. In particular since the shortest string generated by a context free grammar can be exponential in the number of non-terminals of the grammar, even assuming an upper bound on the lengths of the right hand sides of the productions, it is clear that a naive definition of polynomial sample complexity will have counter intuitive implications on the computational complexity. Moreover, since there are cases where there will be exponentially more substrings and constituents than complete strings, algorithms that can operate on substrings have a potential advantage over algorithms that rely on whole strings. We can thus informally distinguish two forms of sample complexity: word complexity, a bound on the number of strings, and letter complexity, a bound on the number of symbols. Given the limited amount of data, we explored a class of algorithms that could exploit this richer source of information.

3. Ideas behind algorithm

We will first outline the intuitions behind the algorithm and then provide a more formal description. The algorithm is based on the idea that we can identify particular substrings that will in general be *constituents*: i.e. derived from a single non-terminal. This relies on the grammar defining a stable relationship between strings and constituency. In general this is not the case, and we can define grammars (representing parity functions, for example) where essentially any string can be a constituent. On the other hand we will find other grammars where the grammatical strings are sparsely distributed, and where a particular substring will generally either be a constituent most of the time or not. An extreme case of this is the class of NTS grammars, defined below, where every substring in the language has the property that either every occurrence is a constituent or no occurrence is. Randomly generated grammars that are deterministic will often have this property, as we argue below. The algorithms we

use exploit this property, and are extended to learn languages where this property is violated slightly. Strictly speaking, we are confusing three separate ideas here: a substring being generated by a non-terminal in a particular derivation of a string; a substring being generated by a non-terminal in every derivation of a particular string; and finally, every occurrence of a substring in the language in any string, being generated by a particular non-terminal.

If a string from Σ^* occurs frequently in the training data, then it may be a constituent—but of course it may also be a substring of a constituent. For a string $w \in$ a language L, and a substring u such that $w = lur$, we will say that u is a constituent in (l, r) if $S \stackrel{*}{\Rightarrow} lXr \stackrel{*}{\Rightarrow} w$ for some $X \in N$, and $S \in R$. In general there may be more than one occurrence of the same substring in a string; in this case some occurrences may be as constituents, and others may not. We will say that u is a constituent if there are strings $l, r \in \Sigma^*$ such that u is a constituent in the context (l, r). For ambiguous context free grammars, this criterion merely requires that for one derivation of a string, the substring is a constituent. of a string to be a constituent is that it is in the yield of some non-terminal. This is not however a sufficient condition.

Example 1. Consider the even palindrome language $P = \{ww^R | w \in \{a, b\}^+\}$, where w^R is the reversal of w and the plus superscript is the Kleene star restricted to at least one occurrence. The obvious grammar for this has one non-terminal S. If we consider the string baabbaabbaab, and the substring baab, we can see that there are three occurrences of the substring *baab*, but only one of these is as a constituent.

Some grammars will have the property that every occurrence of a string in the yield of a non-terminal is a constituent of that type. This property can be neatly characterised in the following way (Boasson & Senizergues, 1985).

Definition 1. A context free grammar $G = (\Sigma, N, S, P)$ is non-terminally separated (NTS) iff for all $X, Y \in N$ and words α, β, γ in $(\Sigma \cup N)^*$ such that $X \stackrel{*}{\Rightarrow} \alpha\beta\gamma$ and $Y \stackrel{*}{\Rightarrow} \beta$ we have $X \stackrel{*}{\Rightarrow} \alpha Y \gamma$.

The origin of the name is clear—if $X \neq Y$ then $y(X) \cap y(Y) = \emptyset$, i.e. the yields of the non-terminals are disjoint.

To explain why these grammars are interesting requires a slight detour into the theory of semi-Thue systems, or reduction systems. The set of rules in a context free grammar can be used in two ways: either to derive sets of strings from a start symbol, or to reduce strings to a single symbol. Consider a rule $X \to \alpha$. This can either be used to rewrite an occurrence of X as the string α as in a normal context free derivation, or it can be used backwards to rewrite the string α as X. In this latter case, the grammar can be considered as a sort of string rewriting system or semi-Thue system. The NTS grammars are precisely those grammars where the set of productions can be used in both ways without changing the set of sentential forms that can be derived from the grammar. While in general many decision problems with Thue systems are undecidable in this case, the NTS property guarantees that we have a Church-Rosser system (Book, 1981), where the word problem (i.e. parsing) can be solved in linear time (Book & Otto, 1993).

These reduction systems are very interesting in the context of grammatical inference (Eyraud, de la Higuera, & Janodet, 2004), since in grammatical inference one is interested in going from the sets of observed strings to the underlying structure; thus a Thue system seems a natural way of modelling this. NTS languages are however rather limited (McNaughton, Narendran & Otto, 1988); whether they are sufficiently powerful depends on the domain. If

the grammar is NTS, then every occurrence of a substring in the yield of a constituent can be rewritten as that constituent. Alternatively, every constituent produced during an exhaustive bottom up parse will be included in the final derivation.

The particular application area that we are concerned with is that of natural language. It is worth pointing out that natural languages do not appear to be strictly NTS. A Biblical quotation, "All we like sheep have gone astray.[1]" has given amusement to generations of schoolchildren because the string "we like sheep" which can be a clause or sentential constituent, is not when it occurs in this sentence.

Of course, not all deterministic grammars are NTS, but randomly generated grammars with a large alphabet are likely to have this property. In practice we found that one of the competition grammars that we were able to identify exactly did have this property and one did not. Deterministic grammars in general will require a large word sample complexity, rather than letter complexity, to allow learnability.

Example 2. Consider the two languages $L_1 = \{a^n b^n c^m | n, m > 0\}$ and $L_2 = \{a^n b^m c^m | n, m > 0\}$. As is well known L_1 and L_2 are deterministic but $L_1 \cup L_2$ is inherently ambiguous and thus non-deterministic. Consider the language $L_3 = dL_1 \cup eL_2$. L_3 is a deterministic language, but neglecting the first symbol it is just $L_1 \cup L_2$. Assume that ambiguous languages are not learnable; this example suggests that any algorithm that can learn all deterministic languages, will have to learn using a strict left-to-right state-merging algorithm rather than a substring based algorithm. L_3 is clearly not NTS.

3.1. Constituent identification

Our approach is therefore based on the idea of identifying substrings that are normally constituents. In some of the grammars that we have here these criteria are not strictly satisfied. We shall discuss the modifications that we made to our algorithm below. To identify whether substrings are constituents, we use a combination of three criteria. First, we only consider strings that occur frequently in the corpus. Secondly, we used an information theoretic criterion based on mutual information (Clark, 2001). Thirdly, we used an idea of substitutability (Harris, 1954), similar to the alignment based learning approach of van Zaanen (2000).

3.1.1. Frequency

The number of times that a substring occurs in a corpus has sometimes been considered to be evidence for constituency. While it is true that substrings that are the yields of non-terminals will be frequent, it is not the case conversely that all frequent strings are constituents, particularly if the distribution of non-terminals is rather unbalanced. Thus while it is possible to define some measure of the extent to which a substring occurs more frequently than chance, this will provide only unreliable information about constituency.

3.1.2. MI criterion

The second of our criteria is an information theoretic criterion based on mutual information. First we define $\Sigma' = \Sigma \cup \{\#\}$, an extended alphabet which consists of the alphabet Σ with an additional boundary symbol #. If we consider a random occurrence of a substring w, we can identify the symbol that occurs before it (which we take to be # if it occurs at the beginning

[1] Most famously used in Handel's "Messiah".

of the string), and similarly the symbol after it (which again we can take to be # if it is at the end of the string). We define the mutual information of the substring w as the mutual information between these two symbols, considered as random variables.

Given a probability distribution over a language $L \subset \Sigma^*$, for any string $w \in \Sigma^*$, define $E[w] = \sum_{u \in \Sigma^*} \sum_{v \in \Sigma^*} Pr(uwv)$. Define $E[\#w] = \sum_{v \in \Sigma^*} Pr(wv)$, and $E[w\#] = \sum_{u \in \Sigma^*} Pr(uw)$, and $E[\#w\#] = Pr(w)$. Note that since there may be more than one occurrence of the substring w in a particular string, we must use the expectations rather than the probabilities.

$$MI(w) = \sum_{l \in \Sigma'} \sum_{r \in \Sigma'} \frac{E[lwr]}{E[w]} \log \frac{E[lwr]E[w]}{E[lw]E[wr]} \tag{1}$$

A simple, but biased, estimator for this quantity is the maximum likelihood estimator which uses the empirical expectation in a finite sample for the expectations in the formula above.

We claim that in many cases constituents will tend to have a high value of this quantity, whereas substrings that are not, will have a very low value. This generalises and improves other information theoretic criterion for constituency which have been proposed in the literature; see for example Lamb (1961) and Stolz (1965) for very early examples of this approach. It has been proposed, for example that constituent boundaries will be characterised by high conditional entropy of adjacent symbols.

Using the notation we introduced above, we can define the right entropy as

$$H_r(w) = \sum_{r \in \Sigma'} -\frac{E[wr]}{E[w]} \log \frac{E[wr]}{E[w]} \tag{2}$$

and the left entropy, H_l similarly.

This is in itself insufficient to identify constituents, but note that since the mutual information between two random variables is bounded by the entropy of each of the two random variables, high entropy of adjacent symbols is necessary for there to be high mutual information between the two symbols.

$$MI(w) \leq \min(H_l(w), H_r(w))$$

We can see the limitations of the entropy based measure by considering the following simple example.

Example 3. Consider a modified palindrome language generated by the grammar $S \to e$, $S \to aSa$, $S \to bSC$, $C \to c_i$ for $i = 1 \ldots N$. Assuming the data is generated from a SCFG, with equal probabilities for the two recursive rules, the following table shows the values of the entropy based measure and the MI based measure.

w	$H_l(w)$	$H_r(w)$	$MI(w)$
ae	$\log 2$	0	0
be	$\log 2$	$\log N$	0
aea	$\log 2$	$\frac{1}{2}(\log 2 + \log N)$	$\log 2$

When N is sufficiently large, strings of the form bS (which are not constituents) will have high right entropy ($\log N$) compared to strings derived from S (which are constituents).

	h	k	r
k	69	14	0
n	0	0	75

Fig. 2 Context counts of common string $qgmt$ from Problem 4. The rows correspond to the symbol occurring before the string, and the columns correspond to the symbol after

Fig. 3 Five most frequent selected substrings containing k b b from Problem 4

String	count	MI	Parse
$mkbbmx$	128	0.69	H_6
$kbbmx$	128	0	-
$mkbbmxws$	51	0.37	H_6 H_5
$kbbmxws$	51	0	-
$tmkbbmx$	30	0.31	t H_6

Indeed, if we consider that the data are generated by a stochastic context free grammar (SCFG), then it can be shown that the mutual information of a string that is derived from a single non-terminal will be bounded by the entropy of the random variable that represents the local context that the non-terminal occurs in. The proof is straightforward, but requires an excessive amount of notation to express. Intuitively, the distribution over $\Sigma' \times \Sigma'$, where Σ' is defined as before to be $\Sigma \cup \{\#\}$, is composed of a linear combination of distributions, which correspond to an occurrence of the non-terminal in the right hand side of a production. Each of these distributions will have zero mutual information, and thus the mutual information of the mixture will be bounded by the entropy of the mixing distribution.

We will now take an example from the Omphalos competition. Figure 2 shows the counts of the contexts for a string that occurs frequently in the training data of Problem 4. We can see that there is a strong correlation between the symbol that occurs on the left and the one that occurs on the right. This indicates that the string is a constituent A that occurs in the right hand side of two rules. One rule will have a right hand side $\ldots nAr \ldots$ and the other $\ldots kA \ldots$. If we examine the correct grammar for this problem we see that indeed there is a rule $H_{16} \rightarrow qgmt$ and two rules $H_6 \rightarrow nH_{16}rqH_{13}xe$ and $H_4 \rightarrow kH_{16}H_4owH_3$. Figure 3 shows the mutual information figures for frequent strings containing a common string kbb. The final column shows whether it occurs as a constituent or sequence of constituents.

3.1.3. Substitutability

Given two strings u and v, if we have $A \stackrel{*}{\Rightarrow} u$ and $A \stackrel{*}{\Rightarrow} v$ then we can be sure that whenever we have a string $xuy \in L$ then $xvy \in L$. This intuitively reasonable assumption is in fact false if the NTS property does not hold. For NTS grammars this is true and for every occurrence of u that is in the yield of A we can substitute it for v and preserve grammaticality. We can thus see that substitutability is an again unreliable piece of evidence for constituency–if two strings are in the yield of the same non-terminal, then they will be substitutable, but the converse is not necessarily true. This insight had also been used as the basis of other learning algorithms (van Zaanen, 2000; Adriaans, Trautwein, & Vervoort, 2000).

In general the substitution relationship alone will not be sufficient to learn the grammar. For any element $\alpha \in V^+$ such that there are strings $l, r \in \Sigma^*$ such that $S \stackrel{*}{\Rightarrow} l\alpha r$, we can see that the set of strings that can be derived from α will all be substitutable with each other in the context l, r. Thus the substitution relationship will also detect sets of strings that can

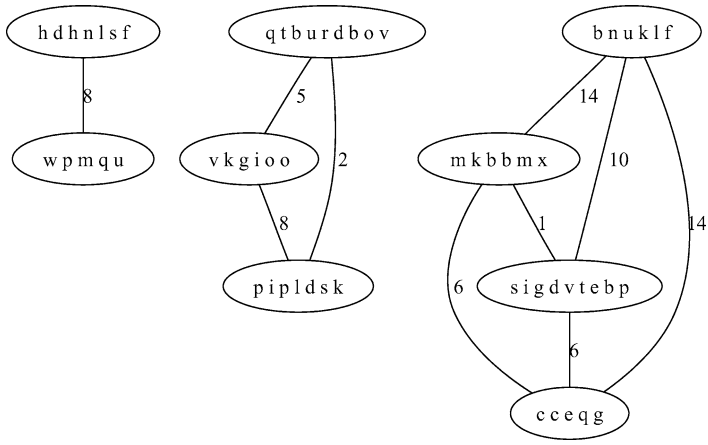

Fig. 4 Part of the substitution graph. Arcs are labelled with the number of pairs of sample strings that are congruent under the rewriting. Each connected component corresponds to a non-terminal in the grammar. In this case these are all complete subgraphs (cliques)

correspond to the yields of strings of non-terminals and terminals, and not just individual non-terminals.

We can use this property to construct what we call a substitution graph. This is an undirected graph, where each node is a substring that may be a constituent. There is an arc between two substrings u and v if there are strings l and r such that both lur and lvr are in the sample. We can further label the arc with the number of such pairs in the sample. Figure 4 shows part of such a graph for Problem 4. If the underlying grammar is NTS, and there is "enough" data, we will normally find that each connected component of the graph is a clique (i.e. a complete subgraph) and will correspond to a subset of the set of yields of a single non-terminal. In practice, given the small data sets, we find that this benign conjunction of properties does not always hold, but as can be seen from Fig. 4, sometimes it does.

One weakness with this approach is that for some grammars, in the worst case, it will need an exponential amount of data to have a high probability of finding a pair of derivation trees that differ by only a single (terminal) local tree.

Example 4. Consider the context free grammar with $\Sigma = \{a, b\}$, $N = \{S, T\}$ and the productions $S \to \underbrace{T \ldots T}_{n}$, and $T \to aa$, $T \to bb$. Assume that the data is generated by a PCFG with probabilities of $1/2$ for each T production. This defines a finite language of size 2^n with each string of equal probability 2^{-n}. Using a birthday attack argument (Yuval, 1979) we can see that we will require of the order of $n^{-1}2^{n/2}$ samples to obtain two derivation trees that differ in only a single production.

3.1.4. Integration

Having constructed the substitution graph, we then remove all nodes that have no arcs leading to them. We also experimented with a further pruning step, which removes all nodes that have a substring that is in a different component. This had the effect of removing components of the graph that correspond to sequences of non-terminals of length greater than 1. It is important to allow substrings to occur in the same component since there may

be rules with center recursion such as $A \to uAv$, $A \to w$ which would lead to a component consisting of w, uwv, $uuwvv \ldots$ and so on.

We then integrate these three selection criteria as follows.

Frequency. We select all substrings from S that occur at least $f_{\min}$ times.
MI filter. Select those substrings whose mutual information is at least $m_{\min}$. This set is C the set of candidate hypotheses.
Substitution graph. Construct substitution graph. Remove nodes of degree 0.

Given the graph, there are a number of heuristics that can be used to select a set of nodes that we hope will correspond to a non-terminal. First, if there is a node corresponding to a single non-terminal in a component, then we can be confident that all nodes joined to that node will be constituents. Alternatively, we can construct a figure of merit based on frequency of strings, number of nodes in the clique, degree of connectivity, and so on. Currently we do not have a principled way of deriving such a figure, nor do the experiments that we have done provide a clear indication of an appropriate formula. Such a formula must depend on the statistical properties of the sample.

3.2. Greedy algorithm

Given a selection of strings corresponding to part of a component of the substitution graph, we must then use this to incrementally construct a grammar that represents the target language.

If the NTS property holds then the ability to identify those frequent substrings that are constituents leads to a natural and efficient algorithm. Since we know that every occurrence of a substring w in the yield of a non-terminal A will be derived from A, we can greedily rewrite the sample strings, to produce a new sample set, a set of sentential forms. We can then repeat the algorithm until we find a small set of very dissimilar sentential forms of reasonable length. These will be the right hand sides of the rules expanding the sentence symbol. Conversely, if we have a large set of very similar sentential forms, it is more likely that these correspond to the application of more than one derivation step to a root symbol. This stopping criterion inevitably uses a certain amount of implicit prior knowledge, derived from the problem description. Algorithms 1 to 3 contain pseudocode that describe this process more precisely.

More formally given a grammar G, and a string or sentential form β we define the greedy generalisation of the string to be

$$\alpha_G(\beta) = \operatorname*{argmin}_{\alpha \in V^* : \alpha \stackrel{*}{\Rightarrow} \beta} |\alpha| \tag{3}$$

Algorithm 1 Pseudocode for LearnCFG(Σ, T)

Require: alphabet: Σ, positive samples $T \subset \Sigma^*$
1: a CFG G
2: G $\leftarrow (\Sigma, \{S\}, \{\}, \{S\})$
3: **repeat**
4: $\quad G \leftarrow$ extendGrammar(G,T)
5: $\quad T \leftarrow$ GreedyReduceData(G,T)
6: **until** $\forall \alpha \in T, G \stackrel{*}{\Rightarrow} \alpha$
7: return G

Algorithm 2 GreedyReduceData(G, T)

Require: grammar G: positive samples $T \subset V^*$
Ensure: reduced data T'
1: $T' \leftarrow \{\}_{multi}$
2: **for all** w in T **do**
3: $\quad T' \leftarrow T' \cup \{\alpha(w, G)\}$
4: **end for**
5: return T'

3.3. Conservative algorithm

If on the other hand, the NTS property does not hold, there are two possibilities that we explored. One method is to proceed as though it does hold, and then to use some later modifications to make the rules more efficient. A better method is to use a more conservative algorithm. We continue to assume that the grammars are deterministic, and thus that there is exactly one derivation for each string in the language. The conservative algorithm only rewrites a string by a non-terminal in certain contexts. This does not mean that it is learning a context-sensitive grammar, but rather that it is applying the context-free rules as reduction rules, but in a constrained way.

Given a positive sample string $w = xuy$, where there is a non terminal $A \stackrel{*}{\Rightarrow} u$, we only rewrite w as xAy when there is another string $w' = xu'y$ such that $A \stackrel{*}{\Rightarrow} u'$, in which case we rewrite both strings. Thus the algorithm only generalises when the generalisation is justified by the set of observed strings.

More formally given a hypothesis grammar $G = (\Sigma, N, P, R)$, where R will often be empty, and two strings u, v drawn from the target language we define the set of generalisations of the pair to be

$$\beta_G(u, v) = \{\alpha \in V^* | \alpha \stackrel{*}{\Rightarrow} u \wedge \alpha \stackrel{*}{\Rightarrow} v\} \tag{4}$$

Algorithm 3 ExtendGrammar(G, T)

Require: grammar $G = (\Sigma, N, P, R)$: positive samples $T \subset V^*$
Ensure: extended grammar G'
1: Construct substitution graph SG;
2: $G' \leftarrow G$
3: **for all** every component C of SG **do**
4: $\quad$ **if** a node of C is a single non-terminal A **then**
5: $\quad\quad$ **for all** nodes α in C except A **do**
6: $\quad\quad\quad$ Add rules of the form $A \rightarrow \alpha$ to G'
7: $\quad\quad$ **end for**
8: $\quad$ **else**
9: $\quad\quad$ create new non-terminal H
10: $\quad\quad$ $N \rightarrow N \cup \{H\}$
11: $\quad\quad$ **for all** node α in C **do**
12: $\quad\quad\quad$ Add rules of the form $H \rightarrow \alpha$ to G'
13: $\quad\quad$ **end for**
14: $\quad$ **end if**
15: **end for**
16: **if** there is a string α in S of length less than l and frequency at least f **then**
17: $\quad$ Add new symbol S_i to R
18: $\quad$ Add rule $S_i \rightarrow \alpha$ to G'
19: **end if**
20: return G'

i.e. the set of all sentential forms that derive both strings. If this set is non-empty, we can define the minimal generalisation $\gamma_G(u, v)$ as the longest element of $\beta_G(u, v)$, i.e. the most specific generalisation of the set; somewhat analogous to the lowest common multiple of two numbers.

$$\gamma_G(u, v) = \underset{\alpha \in \beta_G(u,v)}{\operatorname{argmax}} |\alpha| \tag{5}$$

Given this we can say that a set of strings T is ungeneralisable, when for every distinct pair of strings $u, v \in T$, we have $\beta_G(u, v) = \emptyset$. In the conservative algorithm we minimally generalise every pair of distinct strings until we have a set of sentential forms that is ungeneralisable.

In addition to this modification to the reduction process we also used a different algorithm for constructing the substitution graph. We used a "chart alignment algorithm", to compensate for the fact that the same strings might sometimes be reduced and sometimes not. This takes two different strings $u, v \in V^*$ and finds the shortest pair of strings $x, y \in V^*$ such that there are two strings $w, z \in V^*$ such that $wxz \overset{*}{\Rightarrow} u$ and $wyz \overset{*}{\Rightarrow} v$. This requires extensive use of dynamic programming to find the optimal path through the charts of both u and v.

$$\delta_G(u, v) = \underset{w,z: wxz \overset{*}{\Rightarrow} u \wedge wyz \overset{*}{\Rightarrow} v}{\operatorname{argmin}} |w| + |z| \tag{6}$$

The substitution graph is then constructed from these pairs. Though this algorithm is polynomial, for each pair of strings u, v the algorithm requires $\mathcal{O}(|u|^2|v|^2)$ time, and is also quadratic in the number of strings in the corpus. It was prohibitively expensive to use this for the larger data sets. We are exploring the possibility of various optimisations and approximations.

3.4. Heuristic modifications

Given a grammar that correctly classifies or includes the training data, yet does not give a correct answer on the test data, we can proceed to a series of heuristic modifications to the hypothesised grammars. There are basically two sorts. First, we can generalise the grammar by rewriting the right hand sides of non-terminals. Given two productions $A \to uwv$ and $B \to w$, we can generalise the first by replacing it with $A \to uBv$. Conversely, if we were using a greedy strategy, if we have a production of the form $A \to uBv$ such that for every string in the corpus the occurrence of B in this rule is expanded by a particular rule $B \to w$, then we can make the grammar more specific by replacing the first rule with $A \to uwv$. Note that if the grammar is NTS, we do not have to make this choice–we always select the more general grammar. We constrained this generalisation process by using negative test data where available, but it was not in fact ever useful.

4. Implementation

We started with a set of exploratory data tools, in Perl, that we used to see if the MI heuristic described above would be strong enough to detect non-terminals with this amount of data. We implemented algorithms based on these ideas, prototyping in Prolog (c. 600 lines) and reimplementing in Java (c. 4000 lines).

We experimented with two variants of algorithms based on these ideas: first there is a greedy algorithm that relies on the NTS property, and secondly we have a much slower conservative algorithm that does not. In most cases these algorithms terminated before generating a complete grammar, primarily because of insufficient data. We then had to weaken some of the constraints, until the algorithm would terminate with a grammar that would generate the training data. Finally, we tested on the testing data, and ran generalisation heuristics, testing at each step, until we were successful.

We used a bottom up chart parser of conventional design, and a suffix array with longest common prefix tables (Gusfield, 1997) to efficiently compute all the frequent strings, and to detect alignments. Beyond these, and other standard techniques, we did not use any optimisations. The running times in general were acceptable on a modern workstation, as long as there was sufficient memory to keep all of the charts in memory, and were from a few minutes to few hours depending on the size of the problem.

4.1. Ugly hacks

What we have presented here is a slight idealisation of the process. In the interests of full disclosure we reveal some of the less intellectually respectable techniques we used, given the context of a competition with limited time, data and other resources.

There are a number of free parameters that are used. We could not find a set of parameters that would work for all the problems in the set. We accordingly tweaked the parameters for each size of problem. Frequently the initial parameter settings were too cautious, which meant that the algorithm terminated before producing a grammar that covered the training set. In this case we adjusted the parameters so that the algorithm would continue. With sufficiently large data sets this would not be necessary.

Additionally we manually switched the algorithm between conservative and greedy modes. In general we found that the conservative algorithm was too cautious given the amount of data available. Additionally, it is much slower.

Sometimes the learning algorithm will produce an incorrect rule, or a rule that is too specific. Typically this occurs when we have two pairs of rules $A \to \alpha_1$, $A \to \alpha_2$ and $B \to \beta_1$, $B \to \beta_2$, and we have a rule with right hand side $C \to AB$. In this case we may often find that $\alpha_1\beta_1$ and $\alpha_2\beta_2$ are learnt before the more specific rules for A and B. Thus later, we might learn the rule $C \to AB$ at which point the rules $C \to \alpha_1\beta_1$ and $\alpha_2\beta_2$ become redundant. Avoiding redundancy is important for efficiency; but detecting redundancy in a grammar is undecidable, even for deterministic CFGs since it depends on computing the inclusion relation between different non-terminals in the grammar. We thus used a number of *ad hoc* methods for pruning grammars when this happened.

5. Results

Of the six problems we attempted, we were able to solve all of the small test sets, but only two of the large test sets during the competition. The amount of data was not quite enough to be able to be precise.

Suppose we have a rule $X \to Y_1 Y_2 \ldots Y_n$. Let us further suppose that in every occurrence of this rule in the training data we have that Y_1 is rewritten by the same rule $Y_1 \to \alpha$. Then there are two possible rules we could use that would explain the data: the correct rule or the rule $X \to \alpha Y_2 \ldots Y_n$. There is simply no information in the sample that can help us decide.

If α is a very long and complex string, and we know that the grammars have been randomly generated, then we can use a Bayesian argument to decide whether to generalise or not. However in this case the grammars have been manipulated in a number of ways, and thus we merely have to guess.

After the termination of the competition, the competition organisers verified that the two correct grammars generated exactly the same languages as the target grammars. Moreover, the grammars, while not identical to the target grammars, differed only in a few minor details, some of which were caused by bugs in our code. Figure 5 shows the grammar produced by our system for data set 4. We confirmed with the competition organisors that this was identical to the target grammar.

Fig. 5 Final grammar for data set 4. This was verified as being identical to the target grammar

```
S1 --> d j
S2 --> t e f v c
S3 --> H2 H9
S4 --> H3 H4
S5 --> y H7 H8 H6 H5
S6 --> H7 H8 H6 H5
H3 --> k j n i e
H3 --> w x x o H13 x b  b
H4 --> h o t x d g b
H4 --> k H16 H4 o w H3
H2 --> v d g p n
H2 --> s c t
H2 --> H7 H9 H8 H4 H2 H10 H6 H9 H13
H2 --> H4 H5 H7 H2 H6 H5
H5 --> t f k q v
H5 --> w s
H5 --> H12 H4 H7 H2 H2 H2 H2 H7 H8
H6 --> b n u k l f
H6 --> c c e q g
H6 --> m k b b m x
H6 --> s i g d v t e b p
H6 --> n H16 r q H13 x e
H7 --> f s r w q j
H7 --> p c k t m u o
H8 --> r w t w t
H8 --> s q o k g f s v
H9 --> v p
H9 --> u H10
H10 --> w p m q u
H10 --> h d h n l s f
H12 --> l g d w c r p j
H12 --> h v f p h p t e r
H13 --> p i p l d s k
H13 --> v k g i o o
H13 --> t H10 H12 g g
H13 --> q t b u r d b o v
H16 --> q g m t
H16 --> H6 H2 H7 H12 H2 H12 H5 H2 H8
```

6. Related work

In this section we relate the approach taken here with a variety of other approaches to grammatical induction. First, our algorithm operates on positive data alone, and does not use structural information. In this respect we can contrast it to the algorithm of Sakakibara (1992) which uses derivation trees with labels removed as input. This additional source of information reduces the problem to one much closer to the inference of regular languages (Thatcher, 1967).

We are interested here primarily in classes of algorithms which have some prospect of having provable convergence properties. We can contrast these to methods that perform some non-convex optimisation of an objective function over a whole space of possible grammars. Some approaches have chosen to use a hill climbing algorithm often based on the Expectation-Maximisation algorithm either directly with a stochastic context free grammar (Lari & Young, 1990), which works poorly or using some more sophisticated objective function, for example Klein and Manning (2001) and subsequent papers. Other approaches to non-convex optimisation have also been used, notably evolutionary algorithms (Keller & Lutz, 2005).

Alignment-based learning (ABL) (van Zaanen, 2000) is one algorithm that is based in part on the idea of substitutability. A major difference though is that the alignment algorithm used in ABL can potentially produce more than one substitution per string. To see why this might be a problem consider the language $\{ab^n cd^n e | n > 0\}$. If we align the pair $abcde$ and $abbcdde$ allowing more than one substitution, we would find hypothesized substitutions $b \leftrightarrow bb$ and $d \leftrightarrow dd$, which is incorrect. The correct substitution is $c \leftrightarrow bcd$, which will be found if we restrict the alignment to finding a single edit. The closely related EMILE algorithm (Adriaans, Trautwein, & Vervoort, 2000) also uses the same substitutability heuristic. A more recent algorithm is ADIOS (Solan et al., 2003), which in some respects is similar to the conservative algorithm we presented above, in that it only rewrites in a given context, though a subsequent modification of the ADIOS algorithm uses a "context-sensitive" rewriting rule. ADIOS uses a variety of heuristics to identify constituents: translated into the vocabulary used in this paper, the most important is a conditional entropy constituent likelihood measure, together with a substitutability relationship amongst substrings: i.e. considering u and v to be similar if there are strings l and r of sufficient length, such that both lur and lvr occur as *substrings* in the language, rather than being in the language itself as in our algorithm. The high degree of sparsity in natural language data requires both ABL and ADIOS to use less strict criteria for substitutability.

We can contrast our own earlier approach (Clark, 2001) which used a local notion of context, only the adjoining symbols, together with a statistical measure of distributional similarity as one of the criteria. This could be used to augment the algorithm presented here.

Another point of similarity is with state-merging algorithms for learning deterministic regular languages (Carrasco & Oncina, 1999). In these algorithms the sets of strings that occur as suffixes of a given string are compared. Thus if we assume that the suffix distributions of any two states are disjoint, then the merging algorithm reduces to saying that if we have two strings xuy and xvy, then the states corresponding to xu and xv will be identical. Alternatively, given two sets of suffixes, we say that they are equivalent if their intersection is non-empty. It can be seen that this is a very crude test compared to those normally used in grammatical inference.

We can also compare this to algorithms that only use frequency (Nevill-Manning & Witten, 1997). This algorithm merely generates a compressed representation of the input data. There is no suggestion that the structures it produces will have any relation to the derivation trees of a generating context free grammar.

7. Discussion

There are a number of conclusions and interesting directions for future research that can be identified from the Omphalos competition. First, rather counter-intuitively, larger alphabet sizes in practice can make learning easier, whereas the commonly held view has always been that the large alphabets are intractable; see for example Abe and Warmuth (1992). This can be seen for two reasons. Consider a randomly generated grammar in Greibach normal form. If we keep the number of productions, and the number of non-terminals fixed, but let the size of the terminal alphabet, Σ increase, it is easy to see that for sufficiently large $|\Sigma|$, the grammar is likely to be deterministic, since we are unlikely to have the same letter appearing twice at the beginning of the right hand side of the same rule, and as Σ increases the grammar is likely to be simple deterministic and finally a very simple language. Conversely, take a very simple grammar with a large alphabet, and map the alphabet down to a two letter alphabet randomly. The grammars and languages that result are likely to become more non-deterministic and ambiguous. Thus the tradition in grammatical inference of working with small alphabets, often with only two elements, might have in fact made life more difficult rather than easier. Naturally, this depends on the process by which these grammars are generated, and it is certainly possible to generate random grammars that have particular properties using a specialised process.

The problems in the Omphalos competition are clearly "toy" problems, but it is interesting to contrast the complexity of the tasks with those of some related real world problems. We can compare it on a number of different axes of difficulty, such as the classes of languages, the size of vocabulary, the length of strings, and the size of the grammars as measured in the number of non-terminals and number of productions.

Exact comparisons depend on the application domain. In bioinformatics the range of tasks that are used have vocabularies ranging from 4 (bases) to 20 (amino acids) to many thousands. In natural language, which is our main research interest, we can compare the problems here to unlexicalised grammars which typically use a vocabulary of about 36, with grammars using about 25 non-terminals. However current state-of-the-art grammars used in parsers have substantially larger grammars with thousands of productions. Sentence lengths of naturally occurring language are on average much shorter than the Omphalos data. Thus the Omphalos problems are at least one, and perhaps two orders of magnitude smaller than the main problem classes we are interested in. However we feel that these algorithms should scale smoothly, and we are investigating this empirically.

More importantly, there are a number of areas where the language classes used here are clearly inadequate. First, context free grammars that correctly capture significant grammatical facts, such as agreement phenomena, end up with exponentially large grammars, expressed in a meta-grammar format (Gazdar et al., 1985). Thus formalisms are needed which can express these phenomena in a succinct manner. Secondly, some natural languages are not weakly context free—as is well known, some dialects of Swiss German and some other languages show cross serial dependencies that are beyond the generative capacity of CFGS. These are of the form of $\{a^n b^m c^n d^m | n, m > 0\}$.

With regards to the classes of grammars we have learned in this competition, they have been deterministic.[2] Though natural languages are notoriously ambiguous, it is not clear, ignoring lexical ambiguity, that they are inherently ambiguous. Grammars that have been manually written for natural languages, are always ambiguous, but this is because of a desideratum

[2] We have subsequently solved one of the non-deterministic problems.

that semantic differences (differences in meaning) should be reflected in structurally different syntactic analyses. While this may be desirable from the point of view of semantic interpretation, it is not clear that it is desirable from a learning or processing point of view.

8. Conclusion

We have described the approach we took to solving the Omphalos context free grammar learning competition. This is based on identifying substrings that will normally be constituents. We have discussed a property of grammars, the NTS property that characterises this. We have introduced a novel information theoretic criterion for detecting constituents, and discussed how this can be integrated with other sources of information to produce an efficient and effective learning algorithm for some context free grammars.

Acknowledgments This work was supported by the European Network of Excellence PASCAL and has benefited from a subsidy from the Swiss OFES. We would like to thank the organisers of the Omphalos competition; Brad Starkie, Francois Coste and Menno van Zaanen. We also would like to thank Remi Eyraud and Jean Christophe Janodet for pointing out the literature on NTS grammars. We would also like to thank the reviewers for their very detailed and helpful comments.

References

Abe, N., & Warmuth, M. K. (1992). On the computational complexity of approximating distributions by probabilistic automata. *Machine Learning*, *9*, 205–260.

Adriaans, P., Trautwein, M., & Vervoort, M. (2000). Towards high speed grammar induction on large text corpora. In V. Hlavac, K. G. Jeffery, & J. Wiedermann (Eds.), *SOFSEM 2000: Theory and practice of informatics* (pp. 173–186). Springer Verlag.

Boasson, L., & Senizergues, S. (1985). NTS languages are deterministic and congruential. *J. Comput. Syst. Sci.*, *31*(3), 332–342.

Book, R., & Otto, F. (1993). *String rewriting systems*. Springer Verlag.

Book, R. V. (1981). NTS grammars and Church-Rosser systems. *Information Processing Letters*, *13*(2), 73–76.

Carrasco, R. C., & Oncina, J. (1999). Learning deterministic regular grammars from stochastic samples in polynomial time. *Theoretical Informatics and Applications*, *33*(1), 1–20.

Clark, A. (2001). Unsupervised induction of stochastic context free grammars with distributional clustering. In *Proc. of Conference on Computational Natural Language Learning* (pp. 105–112). Toulouse, France.

Clark, A., & Thollard, F. (2004a). PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, *5*, 473–497.

Clark, A., & Thollard, F. (2004b). Partially distribution-free learning of regular languages from positive samples. In *Proceedings of COLING*. Geneva, Switzerland.

Eyraud, R., de la Higuera, C., & Janodet, J.-C. (2004). Representing languages by learnable rewriting systems. In *International Colloquium on Grammatical Inference*, vol. 3264 of *LNAI* (pp. 139–150). Athens, Greece.

Gazdar, G., Klein, E., Pullum, G., & Sag, I. (1985). *Generalised phrase structure grammar*. Basil Blackwell.

Gusfield, D. (1997). *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press.

Harris, Z. (1954). Distributional structure. In J. A. Fodor & J. J. Katz (Eds.), *The structure of language* (pp. 33–49). Prentice-Hall.

Harrison, M. A. (1978). *Introduction to formal language theory*. Addison Wesley.

Kearns, M., & Valiant, G. (1994). Cryptographic limitations on learning Boolean formulae and finite automata. *JACM*, *41*(1), 67–95.

Keller, B., & Lutz, R. (2005). Evolutionary induction of stochastic context free grammars. *Pattern Recognition*. to appear.

Klein, D., & Manning, C. (2001). Distributional phrase structure induction. In *Proceedings of CoNLL 2001* (pp. 113–121).

Lamb, S. M. (1961). On the mechanisation of syntactic analysis. In *1961 Conference on Machine Translation of Languages and Applied Language Analysis*, vol. 2 of *National Physical Laboratory Symposium No. 13* (pp. 674–685). London: Her Majesty's Stationery Office.

Lari, K., & Young, S. J. (1990). The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, *4*, 35–56.

McNaughton, R., Narendran, P., & Otto, F. (1988). Church-Rosser Thue systems and formal languages. *J. ACM*, *35*(2), 324–344.

Nevill-Manning, C. G., & Witten, I. H. (1997). Identifying hierarchical structure in sequences. *Journal of Artificial Intelligence Research*, *7*, 67–82.

Ron, D., Singer, Y., & Tishby, N. (1995). On the learnability and usage of acyclic probabilistic finite automata. In *COLT 1995* (pp. 31–40). Santa Cruz CA USA.

Sakakibara, Y. (1992). Efficient learning of context free grammars from positive structural samples. *Inf Computing*, *97*(1), 23–60.

Solan, Z., Horn, D., Ruppin, E., & Edelman, S. (2003). Unsupervised context sensitive language acquisition from a large corpus. In *Proceedings of NIPS-2003*.

Starkie, B., Coste, F., & van Zaanen, M. (2004). The Omphalos context-free grammar learning competition. In *International Colloquium on Grammatical Inference*, vol. 3264 of *LNAI* (pp. 16–27). Athens, Greece.

Stolz, W. (1965). A probabilistic procedure for grouping words into phrases. *Language and Speech*, *8*(4), 219–235.

Thatcher, J. W. (1967). Characterizing derivation trees of context free grammars through a generalization of finite automata theory. *Journal of Computer and System Sciences*, *1*(4), 317–322.

van Zaanen, M. (2000). ABL: Alignment-based learning. In *COLING 2000—Proceedings of the 18th International Conference on Computational Linguistics*.

Yuval, G. (1979). How to swindle Rabin. *Cryptologia*, *3*, 187–189.