l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

UNIVERSITÉ DE NANTES

Thèse de Doctorat

James SCICLUNA

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
Label européen
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire d'Informatique de Nantes-Atlantique (LINA)

Soutenue le 1 décembre 2014

Grammatical Inference of Probabilistic Context-Free Grammars

JURY

Rapporteurs :	**M. Alexander CLARK**, Lecturer, King's College, London
	M. Amaury HABRARD, Professeur, Université de Saint-Etienne
Examinateurs :	**M. François COSTE**, CR1, INRIA, Rennes
	M. Jean-Christophe JANODET, Professeur, Université d'Evry
Directeur de thèse :	**M. Colin DE LA HIGUERA**, Professeur, Université de Nantes

Remerciements

I would like to thank first of all my supervisor Colin de la Higuera. The discussions I had with him and the insights and ideas he shared with me were crucial for this thesis. His constant support made everything easier. Given his knowledge of the subject area and his pedagogical way of explaining things, I couldn't have asked more from a supervisor. On a lighter note, I also appreciate the fact that he shielded me away from boring bureaucratic stuff so that I could focus solely on the nice research problems.
I would like to thank Alex Clark, Amaury Habrard, François Coste and Jean-Christone Janodet for kindly accepting to be part of the jury. Thanks also goes to Jean-Christophe Janodet and Jérémie Bourdon for being part of the *comité de thèse*.
I would also like to thank Menno Van Zaanen for my short visit to Tilburg and Mark-Jan Nederhof for collaborating with us on one of our contributions in this thesis.
I'd like to thank my fellow lab mates from LINA. Thanks for making my stay in Nantes more pleasurable and thanks for putting up with my lack of French! Obviously, you're always welcome to visit Malta anytime.
A special thanks goes to my family, friends and my girlfriend Christine back in Malta. *Grazzi Chri tal-paċenzja li ħadt bija u skużani ta kemm tertaqtlek qalbek. Insomma, wara ħafna dagħa u tkaxkir tas-saqajn, spiċċajt lestejtha wkoll din l-ostja teżi. Issa waħda ħara Paceville kif tmiss il-liġi!*

Contents

1

Introduction

Machine learning is a subfield of artificial intelligence which seeks to answer the scientific question of how we can build computer systems that are able to *learn*. Analysing these systems and trying to find general laws which govern learning processes is also of interest to the field. We use the definition given by (Mitchell, 1997) to describe what it means when we say that a computer system is able to learn:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"

The utility and applications of such computer systems are vast. Such systems are already being used nowadays in a variety of areas such as natural language processing, bioinformatics, robotics, stock market analysis and others. Basically, wherever there is data available (e.g. corpora of natural language text, DNA sequences, sensory data, historical stock market data) from which information can be extracted that can be useful for some task (e.g. building language models, prediction of protein structures, robot learning, stock market predictions), then machine learning systems can be applied. Machine learning is becoming evermore relevant today because of the emergence of the Big Data phenomenon. In many fields, very large data sets are available and one major challenge is to make best use of this data for modelling or predicting natural world phenomena and automating tasks for which a direct 'non-learning' algorithm is very difficult to implement. There is a lot of untapped potential in machine learning given that a lot of this data is still not analysed and unstructured. So clearly, all this is a huge motivation for building and studying learning systems.

1.1 Grammatical Inference

One branch of machine learning is Grammatical Inference (GI). In GI the problem is to learn formal language representations from information about an unknown target formal language. Formal languages are simply (possibly infinite) sets of strings and a representation of a formal language L is any finite model which is able to generate or exactly describe L. Typical formal language representations take the form of grammars or automata. Therefore, the hypothesis space in GI is most often some class of grammars or automata (e.g. regular automata or context-free grammars) which contains all the possible models that can be learned. The information given (from which learning takes place) can be of various forms, such as from:

– *Text*: A finite sample of strings in the target language
– *Informant*: A finite sample of strings labelled as positive or negative. Positive strings are contained in the target language and negative strings are not.
– *Structured Data*: A finite sample of strings with added structural descriptions assigned by the target grammar or automaton.
– *Queries*: Access to an oracle which is able to answer queries posed by the learning algorithm. There are several forms of queries, including for example membership queries (where the learner asks whether a particular string is contained in the target language) and equivalence queries (where the learner asks whether a hypothesized grammar or automaton is equivalent to the target one).
– *Statistical Evidence*: In case the target grammar or automaton is of a probabilistic nature, then it is normally assumed that the data given is generated under the i.i.d (independent and identically distributed) assumption. With this assumption, statistical evidence from the given sample serves as an important source of information for the learning algorithm.

GI can be applied wherever grammars or automata serve as good models for some task. For example, in Natural Language Processing (NLP), Probabilistic Context-Free Grammars (PCFGs) serve as good models for assigning probabilities to sentences and describing the structure of sentences (where these tasks can prove very useful for speech recognition, machine translation and other NLP related tasks). Thus, GI can be used to induce such grammars from corpora of unstructured raw natural language sentences. Another example is Bioinformatics, where Hidden Markov Models (HMMs) and PCFGs have proven to be good descriptors for DNA, RNA and protein structures. Practical GI learning algorithms that infer finite state automata include (Lang et al., 1998; Adriaans and Jacobs, 2006; Coste and Nicolas, 1998; Heule and Verwer, 2010), and for the context-free case (Sakakibara et al., 1994b; Nakamura and Ishiwata, 2000; Stolcke, 1994; Petasis et al., 2004; Adriaans et al., 2000; van Zaanen, 2001; Clark, 2001).

Apart from applications in various fields, an integral part of the work done in GI is concerned with the development of theoretical learning algorithms. These algorithms provably learn (given some formal definition of what learning means) classes of grammars or automata in well-defined scenarios (where the information given and conditions which this information must satisfy are clearly defined). Such algorithms offer guarantees and insights that lack in practical algorithms. Moreover, such algorithms can serve as the foundation of more robust practical algorithms. The following are examples of theoretical learning algorithms that induce regular automata from text (Angluin, 1982; García et al., 1990), an informant (Gold, 1978; Oncina and García, 1992) and queries (Angluin, 1987). The following are text (Clark and Eyraud, 2007; Yokomori, 2003; Laxminarayana and Nagaraja, 2003), informed (Sempere and García, 1994; Higuera and Oncina, 2002; Eyraud et al., 2007) and query (Sakakibara, 1991; Clark, 2010a) learners that induce context-free grammars. Examples of learners which induce probabilistic regular automata and probabilistic context-free grammars are (Carrasco and Oncina, 1994a; Clark and Thollard, 2004; Ron et al., 1995) and (Higuera and Oncina, 2003) respectively.

Other more fundamental and theoretical works in GI prove general learnability results or limitations on what can be learnable within a particular learning model. Important works such as (Gold, 1967; Angluin, 1988a,b; Kearns and Valiant, 1994; Higuera, 1997) fall in this category. Questions on certain properties of grammars or automata themselves can also be of interest to grammatical inference (Fernau and Higuera, 2004). Normal forms, equivalence or distances between models and the computability of functions on grammars and automata can all be of help.

The reader is referred to (Higuera, 2010) for a comprehensive review of the grammatical inference field.

1.2 Three Components of Grammatical Inference Algorithms

We identify the following as three important components of any GI learning algorithm:

1. Restricted Target Class of Grammars
2. Learning Strategy
3. Preference Bias

1. *Restricted Target Class of Grammars*: Let **A** be a GI algorithm that learns languages described by grammars in a class **C** of grammars. This means that for any language L described by a grammar $G \in \mathbf{C}$, **A** is capable of learning a grammar G' generating L. The hypothesis space is the class $\mathbf{C}'$ of grammars containing all the possible grammars G' that **A** can return. On the other hand, the class **C** is the target class of grammars. The classes **C** and $\mathbf{C}'$ may or may not coincide. It is usually the case in GI that grammars in $\mathbf{C}'$ are normal form grammars for those in **C**. For a target class **C** to be learnable, it normally needs to be defined as a restricted version of some general form of grammars (since these are normally not learnable under a variety of learning models). The type of restriction used is of importance to its learnability. This is why we include it as one of the key components of any GI algorithm.

   Restrictions on the target class of grammars can take various forms. Some use syntactic restrictions on grammar classes (Higuera and Oncina, 2002; Yokomori, 2003; Eyraud et al., 2007). Others use purely language-theoretic restrictions (García et al., 1990; Rogers et al., 2010), whilst some give both syntactic and language-theoretic characterizations (Angluin, 1982; Sempere and García, 1994). An effective approach is to impose conditions on the grammars such that their features are directly related to features of their language (Clark and Eyraud, 2007; Clark, 2010a; Yoshinaka, 2010). The idea here is to facilitate the learning process, since features of the target language can be observable from the information given and thus used to easily find the features of the grammars (that are not directly observable from the given information). In the case of practical GI algorithms, the target class of grammars is sometimes not known, not well-defined or difficult to find (Petasis et al., 2004; van Zaanen, 2001; Solan et al., 2005).

2. *Learning strategy*: Some form of strategy has to be taken by a GI algorithm in order to generalize from the information given. There is a dichotomy between methods which start from a very general hypothesis and take decisions that specialize this hypothesis (normally through state-splitting, see (Tellier, 2007; Belz, 2002; Klein and Manning, 2002) as examples), and methods which start from an overly-specific hypothesis and take decisions that generalize this hypothesis (normally through state-merging, see (Oncina and García, 1992; Sakakibara, 2005; Adriaans et al., 2000) as examples). The latter is the most common strategy used. Decisions are normally either based on patterns found in the data or are taken so as to maximize/minimize some optimisation function on grammars (or a combination of both). Decisions are normally taken greedily due to the exponential nature of the search space. The nature of the decisions and how they are taken form the learning strategy.

3. *Preference bias*: Another important component of any GI algorithm is the preference bias. This is the criterion used by the algorithm to choose between competing hypotheses that are consistent with the information given. Theoretical GI algorithms are normally designed to return hypotheses in a particular normal-form (Angluin, 1982; Sempere and García, 1994), which in some cases correspond exactly to the smallest hypothesis (where the size of the hypothesis is typically defined in terms of the number of states). Practical algorithms tend to explicitly (Adriaans and Jacobs, 2006; Petasis et al., 2004; Langley and Stromsten, 2000) or implicitly (Lang et al., 1998) favour small representations.

For some practical algorithms, the preference bias used is not known or not well understood (Solan et al., 2005; Adriaans et al., 2000). An interesting comparison can be made between techniques which take into consideration the preference bias in their learning strategy and others which separate the two.

1.3 Grammatical Inference of Probabilistic Context-Free Grammars from Text

Whenever a sample of strings is available, and a distribution over such strings or a distribution over tree structures of these strings needs to be inferred, then Probabilistic Context-Free Grammar (PCFG) learning from text can be used for this task. This scenario can be found in a variety of settings, especially where tree structures on strings have some important meaning (for example, parse trees in NLP or RNA structures in Bio-informatics). The problem is that not many GI algorithms that learn PCFGs from text are found in the literature. This is mainly because it is a difficult task. We can identify the following three main difficulties that make learning PCFGs a difficult task:

1. If we take the strategy where we first learn the underlying structure of a PCFG (i.e. learn a CFG) and then assign probabilities to this structure, then we are faced with the known problems of learning CFGs. CFG learning is well-recognized as a difficult problem. Eyraud (2006) and Higuera (2010) identify several reasons for this, namely:
   - The fact that important questions on CFGs, such as whether two CFGs are equivalent or whether one CFG generates a language included in the language of another CFG, are undecidable.
   - There is no direct relationship between the states of a CFG and its language. In other words, no Myhill-Nerode type of theorem for CFGs (which proves to be very helpful for learning regular grammars).
   - There are CFGs of size n which generate strings of size at least exponential in n. This is problematic if we want a learner which is capable of working within polynomial bounds w.r.t. the size of the grammar (Higuera, 2010).
   - A CFG in general cannot be built as a collection of different independent sub-parts. The dependency between rules can be complex and removing or adding one rule to the grammar can change its language drastically. All this makes it difficult to have a learner which uses a divide and conquer type of strategy.

2. Many CFG learning algorithms aim to induce grammars which generate the same language as the target grammar. This means that the learned grammar can be structurally very different from the target one whilst still being a perfectly correct grammar to learn as long as it generates the target language. In this case, learning the structure of the target grammar is not necessary. However, for PCFG learning, the structure of the target grammar is of importance and has to be taken into consideration in the learning algorithm. This is because equivalent PCFGs tend to have similar structures. Also, given two equivalent CFGs G_1 and G_2 which are structurally very different, it may very well be the case that there exists no two probability assignments θ_1 and θ_2 such that the PCFG (G_1, θ_1) is equivalent to the PCFG (G_2, θ_2) (we give an example of this in Section 4.4). The problem is that learning structure from text (i.e. without even one example of a correctly structured string) is a difficult task simply because of the inherent lack of structural information.

3. There are still some basic open questions regarding PCFGs. For example, it is not known whether the equivalence between two PCFGs is decidable or not. Distances between PCFGs have barely been

investigated and we know of no approximation algorithm for these distances. An interesting question (which to our knowledge has not been tackled) is whether there exists some type of normal form for CFGs such that if two equivalent CFGs G_1 and G_2 are in this form, then there must exist probability assignments θ_1 and θ_2 such that PCFG (G_1, θ_1) is equivalent to PCFG (G_2, θ_2). The lack of these type of results on PCFGs makes the learning task more difficult.

Much of the work done on PCFG learning consists of practical GI algorithms which are mostly aimed at specific applications. The two main applications are in NLP (for unsupervised parsing (Johnson, 1998; Klein, 2004) and language modelling (Jurafsky et al., 1995; Benedí and Sánchez, 2005)) and in Bioinformatics (Sakakibara et al., 1994b; Salvador and Benedi, 2002).

PCFG learning can take the form of either parameter estimation or structure learning. In parameter estimation, a context-free grammar is fixed and the focus is on assigning probabilities to this grammar using Bayesian methods (Johnson et al., 2007), maximum likelihood estimation (Lari and Young, 1990) or other techniques (Cohen and Smith, 2010b; Cohen et al., 2012). In structure learning, the focus is on building the right grammar rules from scratch. The search for the correct structure is normally guided by distributional information (van Zaanen, 2001) or by the use of a bias in favour of small grammars that compactly describe the sample (Stolcke, 1994) (more information on these techniques can be found in Sections 3.2.3 and 3.2.2 respectively). Whenever a-priori knowledge is available on the structure of the target grammar, then parameter search would normally be the best approach to take. However, in a more general setting where only a sample is available and no knowledge of the type of structure to be learned, then the more difficult structure learning approach has to be taken. Throughout this thesis, we will be exclusively focusing on the latter approach.

1.4 Contributions

1.4.1 Major Contribution

PCFG structure learning from text is a difficult and widely open problem, with very few recent theoretical and practical results (Higuera, 2010). However, such a task can be helpful in a variety of fields (Higuera, 2010). This is because there are a number of real world problems for which a lot of unstructured data is available (e.g. 'raw' natural language sentences, DNA sequences, etc.) which needs to be modelled and assigned structure automatically for a variety of reasons (e.g. semantic interpretation for natural language sentences, RNA structure prediction etc.). PCFGs can serve as good models for such tasks. Moreover, unsupervised learning of PCFGs is an interesting problem on its own since it can serve as an example of how far learnability is possible from the most basic possible information given. All this serves as motivation for our major contribution in this thesis: A practical PCFG learning algorithm, based on a principled approach and with some proven properties, that has been applied for two important tasks in NLP, unsupervised parsing and language modelling.

We earlier identified three key components of any GI algorithm. Our learning algorithm tackles each as follows:

1. *Restricted Target Class of Grammars*: We define a new subclass of PCFGs which is very similar to the one defined in (Clark, 2010a). The states (i.e. non-terminals) of grammars in this class are directly related to certain classes of substrings in the sample given. Evidence for inferring these classes of substrings can be found in the sample itself (through the occurrence of certain patterns in the sample). Therefore, with our restricted class of grammars, the learning problem is reduced to that of finding these classes of substrings for which evidence is available. This overcomes the problem with CFGs

in general where no direct relationship exists between the non-terminals and strings in the language. Although the relationship between non-terminal and substrings for our class is not as powerful as the relationship between the DFA states and prefixes used in DFA learning (using the Myhill-Nerode theorem), it still simplifies the learning problem compared to general CFG learning.

2. *Learning Strategy*: Due to the restricted target class of grammars, the learning problem in our algorithm is reduced to that of identifying classes of substrings. We start from the most specific hypothesis (where every substring is placed in a class of its own) and then generalize from this using one simple generalization operator: placing pairs of substrings in the same class (i.e. merging the non-terminal classes of substring). Decisions on which substrings should be grouped are taken based on the contexts in which the substrings are found in the sample. Decisions are taken greedily, so whenever two substrings are grouped, they are never then placed back into separate classes.

3. *Preference Bias*: With the classes of substrings in hand, there will still be a huge number of grammars consistent with these classes. This is because:

   (a) There are many ways of choosing which classes of substrings correspond to non-terminals of the grammar to be learned.

   (b) There is a huge number of possible ways of choosing which relationships between these classes should form the production rules of the grammar to be learned.

   Our algorithm solves this issue by trying to find the smallest possible number of classes and relationships between these classes to form the smallest possible grammar. Thus, our bias is in favour of smaller grammars. We do this by reducing the problem to the well-known *Minimum Satisfiability* (MIN-SAT) NP-Complete problem. Therefore, we can make use of sophisticated solvers for this problem.

Note that the approach we take is different from the approach normally taken by GI algorithms in general. Many GI algorithms incrementally build improved hypothesis grammars until they reach a point where, according to some metric, the grammar is deemed to be good enough as the final hypothesis grammar. This means that the preference bias is taken into consideration along the learning process. However in our case, we separate the learning process from the preference bias. Our learning process consists only of a clustering procedure on the substrings in the sample (without the need of building hypothesis grammars or even considering the bias in favour of small grammars in the process). From the clusters, we then simply reduce the problem to that of finding the smallest possible hypothesis grammar (i.e. we reduce the problem to a combinatorial one). The advantage of such an approach is that it divides the whole problem into two well-defined sub-problems: finding the target classes of substrings and finding the smallest possible grammar.

We tested our system on samples generated from typical context-free languages and artificial natural language grammars. In general, our system was capable of inferring grammars which were structurally similar to the target grammars and also generate probabilistic languages close to the target languages. We also tested our system on a natural language corpus. Although the results obtained on the corpus are worse than those obtained by state-of-the-art systems, we show that there exists a grammar consistent with the classes of substrings learned, which generates tree structures close to the target treebank structures.

1.4.2 Minor Contribution

We mentioned earlier that one of the problems with PCFG learning is the fact that there are still some basic open questions regarding PCFGs. For example, questions on distances between PCFGs have not been thoroughly investigated and the question of whether PCFG equivalence is decidable or not is still open. It

can be very helpful for PCFG inference if we know how to check if two PCFGs are equivalent or have an algorithm that calculates the distance between two PCFGs or at least is able to find an approximation for the true distance. Having such algorithms is helpful for comparing hypothesis grammars with target grammars as part of the evaluation process. Moreover, key grammatical inference operations such as state or non-terminal merging are often determined by the equivalence or distances between the states, and so equivalence or distance (approximation) functions can be helpful in the learning process itself.

Given the importance of such questions and the lack of work done on investigating them, we give as a minor contribution in this thesis the following three results:

1. We show that a good number of distances between probability distributions are uncomputable for PCFGs (i.e. given two PCFGs, it is not possible to calculate the distance between the probability distributions generated by these grammars). This is a negative result and is of no practical use for grammatical inference apart from highlighting the need of approximation algorithms for distances between PCFGs.
2. We prove that the problem of determining whether two PCFGs are probabilistically equivalent is interreducible to the problem of finding Chebyshev distance between two PCFGs. Thus, this result gives a new insight into the open problem of whether the equivalence problem for PCFGs is decidable or not.
3. We show that the consensus string problem (i.e. the problem of finding the most probable string) for PCFGs is computable (yet the algorithm we give is intractable).

1.4.3 Publications

As explained earlier, the major contribution in this thesis is a practical PCFG learning algorithm with some proven properties that has been applied for unsupervised parsing and language modelling. The application of this algorithm is described in the paper "PCFG Induction for Unsupervised Parsing and Language Modelling" authored by James Scicluna and Colin de la Higuera, which will appear in the proceedings of the EMNLP 2014 conference. A similar paper by the same authors was presented at CAp'2014 (French Machine Learning Conference). A more theoretical view of this algorithm is given in the paper "Grammatical Inference of some Probabilistic Context-Free Grammars from Positive Data using Minimum Satisfiability" authored by James Scicluna and Colin de la Higuera, which appears in the JMLR Workshop and Conference Proceedings and was presented at the ICGI 2014 conference (where it won the best student paper prize).

Our minor contribution consists of a compendium of undecidability results for distances between PCFGs along with two positive results. This material can be found in an article entitled "On the Computation of Distances for Probabilistic Context-Free Grammars" authored by Colin de la Higuera, James Scicluna and Mark-Jan Nederhof, which is archived on ArXiv.

1.5 Outline of the Thesis

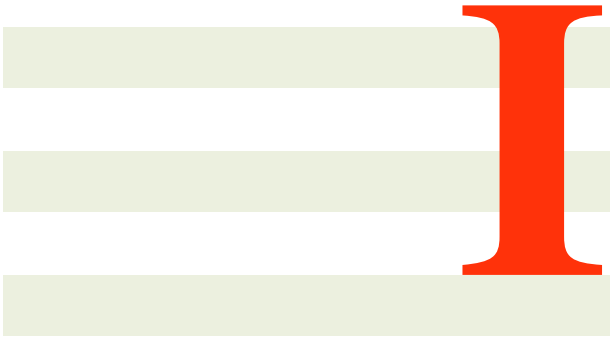
The thesis is structured in a fairly straight-forward manner. It is made up of two parts. The first part contains the background material required for one to understand the two principle components of our work, namely:

1. PCFGs, which are the models we are interested in. A comprehensive review of PCFGs is given in Chapter 2.
2. Grammatical Inference, which is our field of interest. A review of this field is given in Chapter 3.

The second part of the thesis contains our contributions. Our minor contribution is explained in Chapter 4. Our major contribution is explained in 3 chapters, one chapter for each component of the learning algorithm. The components correspond to the ones explained earlier in Sections 1.2 and 1.4.1. Therefore:

1. In Chapter 5, we define and analyse a restricted target class of PCFGs which is 'designed' to be learnable.
2. In Chapter 6, we explain the learning strategy our algorithm takes in order to generalize from the given data.
3. In Chapter 7, we show how our algorithm prefers smaller grammars when choosing between competing hypothesis grammars.

In Chapter 8, we evaluate our major contribution through experiments on typical context-free grammars, artificial natural language grammars and natural language corpora. Finally, some concluding remarks are given in Chapter 9.



Background

2

Probabilistic Context-Free Grammars

Probabilistic Context-Free Grammars (PCFGs) are statistical models based on context-free grammars, which are a well known type of formal grammars in formal language theory. The main attractive feature of PCFGs is that they describe probability distributions on strings and more importantly on tree structures of the same strings. This makes them suitable for structured prediction in a variety of areas where tree-like structures on strings have some meaning.

In this chapter, we formally define CFGs and PCFGs (Section 2.1) and explain how one can decide whether a given PCFG actually describes a distribution on strings (Section 2.2). We also describe the following important algorithms related to PCFGs:

- Checking if a particular string is generated by a given PCFG (Section 2.3)
- Computing the probability assigned by a PCFG to a string (Section 2.3)
- Finding the most probable parse tree for a string (Section 2.3)
- Computing the probability that a PCFG generates strings starting with a given prefix (Section 2.4)
- Estimating the unknown probabilities of a PCFG from a sample of strings (Section 2.5)

We conclude with a brief review of the relevant literature (Section 2.6).

2.1 Definitions

2.1.1 Defining Context-Free Grammars

A *context-free grammar (CFG)* is a tuple $\langle N, \Sigma, R, I \rangle$, where N is a set of non-terminal symbols, Σ is a set of terminal symbols (where $N \cap \Sigma = \emptyset$), $R \subseteq N \times (N \cup \Sigma)^*$ is a set of production rules and $I \subseteq N$ is a set of starting non-terminals. Unless otherwise specified, we use a, b, c, d for symbols in Σ, other lower-case Roman letters for strings in Σ^*, capital Roman letters for non-terminals (apart from G which will normally be used to denote a CFG), S for any starting non-terminal and Greek letters for strings in $(N \cup \Sigma)^*$. A production rule (A, α) of a CFG is written as $A \to \alpha$, where A is the left-hand side (LHS) of the rule and α is the right-hand side (RHS). Multiple production rules with the same RHS: $A \to \alpha_1$, $A \to \alpha_2$, $\ldots A \to \alpha_n$, can be written as follows: $A \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$. Usually, only the production rules of a CFG are written and the non-terminals, terminals and starting non-terminals are induced from the rules themselves according to

the notation defined above. For example, the CFG $\langle N, \Sigma, R, I \rangle$ where $N = \{S_1, S_2, C, D\}$, $\Sigma = \{a, b, c, d\}$, $R = \{(S_1, aS_1b), (S_1, ab), (S_2, CD), (C, cC), (C, c), (D, dD), (D, d)\}$ and $I = \{S_1, S_2\}$ is simply written as follows:

$$S_1 \to aS_1b \mid ab$$
$$S_2 \to CD$$
$$C \to cC \mid c$$
$$D \to dD \mid d$$

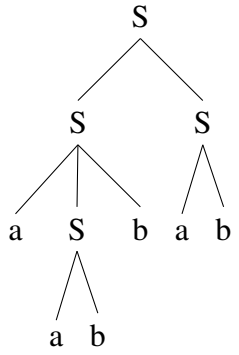
The *derivational relation* over a CFG G, denoted as $\Rightarrow_G$, is a relation on $(\Sigma \cup N)^*$ defined as follows:

$$\alpha \Rightarrow_G \beta \text{ if } \exists A \in N, \exists \gamma, \mu, \omega \in (\Sigma \cup N)^* : (\alpha = \mu A \omega) \textit{ and } (\beta = \mu \gamma \omega) \textit{ and } (A \to \gamma \in R)$$

The transitive closure of $\Rightarrow_G$ is denoted as $\overset{+}{\Rightarrow}_G$ and the reflexive transitive closure of $\Rightarrow_G$ is denoted as $\overset{*}{\Rightarrow}_G$. The subscript is omitted whenever the grammar G is clear from the context. A string $\alpha \in (N \cup \Sigma)^*$ derives another string $\beta \in (N \cup \Sigma)^*$ if $\alpha \overset{*}{\Rightarrow}_G \beta$. A string $\alpha \in (N \cup \Sigma)^*$ reaches another string $\beta \in (N \cup \Sigma)^*$ if for any $\mu, \omega \in (N \cup \Sigma)^*$, $\alpha \overset{*}{\Rightarrow}_G \mu\beta\omega$. A grammar G derives (resp. reaches) a string $\alpha \in (N \cup \Sigma)^*$ if for any $S \in I$, S derives (resp. reaches) α.

The *language* generated from a non-terminal A is $L(A) = \{w \in \Sigma^* | A \overset{*}{\Rightarrow} w\}$ and the *sentential forms* generated from a non-terminal A are $\hat{L}(A) = \{\alpha \in (\Sigma \cup N)^* | A \overset{*}{\Rightarrow} \alpha\}$. The *language* generated by a CFG G is $L(G) = \bigcup_{S \in I} L(S)$ and the *sentential forms* of a grammar G are $\hat{L}(G) = \bigcup_{S \in I} \hat{L}(S)$. A *context-free language* (CFL) is any language generated by a CFG. Two CFGs are *equivalent* if they generate the same language.

A *derivation* of a non-terminal A, denoted D_A, is a sequence of length n of elements in $(N \cup \Sigma^*)$ such that $D_A[1] = A$, $D_A[n] \in \Sigma^*$ and for every $i \in 1 \ldots n-1$, $D_A[i] \Rightarrow_G D_A[i+1]$. A derivation of a grammar G is any D_S for $S \in I$. The subscript of D is omitted whenever we refer to a derivation in general (from any non-terminal). A *leftmost derivation* is a derivation D of length n such that for every $i \in 1 \ldots n-1$, the leftmost non-terminal of $D[i]$ is the one substituted with the RHS of a production rule to obtain $D[i+1]$. The set of leftmost derivations of a string $w \in \Sigma^*$ from a grammar G (resp. non-terminal A), denoted $LD_G(w)$ (resp. $LD_A(w)$), is made up of all the leftmost derivations of G (resp. A) of length n where $D[n] = w$. Every leftmost derivation can be graphically represented as a parse tree. For example, the following is the parse tree for the leftmost derivation $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabbS \Rightarrow aabbab$:



For any derivation D of length n, $r(D)$ is a sequence of length $n-1$ of the production rules used in the derivation, where $r(D)[i]$ is the rule used to reduce $D[i]$ to $D[i+1]$. So, for the derivation D = $S \Rightarrow SS \Rightarrow aSbS \Rightarrow aabbS \Rightarrow aabbab$, $r(D)$ is $[S \to SS, S \to aSb, S \to ab, S \to ab]$.

The *degree of ambiguity* of a string $w \in \Sigma^*$ with respect to a grammar G is the number of leftmost derivations of w from G. A CFG G is said to be *ambiguous* if at least one string has a degree of ambiguity bigger than 1. A CFL L is said to be *inherently ambiguous* if no unambiguous CFG is capable of generating L.

A context-free grammar is *proper* if it satisfies the following three conditions:

1. It is cycle-free, i.e. no non-terminal A exists such that $A \overset{+}{\Rightarrow} A$.

2. It is λ-free, i.e. either no rules with λ on the RHS exist or exactly one exists with S on the LHS (i.e. $S \to \lambda$) and S does not appear on the RHS of any other rule.
3. It contains no useless symbols or non-terminals. This means that every terminal and non-terminal should be reachable from S and every non-terminal should derive at least one string from Σ^*.

For every CFG G, there exists an equivalent proper CFG G' (Harrison, 1978). A grammar is in *Chomsky-Normal Form* (CNF) if every production rule is in $(N \times N^2)$ or $(N \times \Sigma)$. For every CFG G, there exists an equivalent CFG G' in CNF (Harrison, 1978). An equivalent CFG in CNF can be built in polynomial time from any CFG following these steps (Rich, 2008):

1. Transform all production rules which have more than two symbols in the RHS into rules with exactly two symbols in the RHS. This is done by applying the following simple transformation until all rules have at most two symbols on the RHS:

   Transform $A \to X_1 X_2 \dots X_n$ into $A \to X_1 A'$ and $A' \to X_2 \dots X_n$, where A' is a new non-terminal

2. Remove all production rules that have the empty string λ as the RHS (apart from rules with the starting non-terminal on the LHS). For every $A \to \lambda$ which was removed and for every rule $B \to \alpha A \beta$, add rule $B \to \alpha\beta$. If a removal of a rule $A \to \lambda$ results in a grammar which has no rule with A on the LHS, remove also any rule where A occurs in its RHS.
3. For every two non-terminals A, B such that $A \overset{*}{\Rightarrow} B$ *and* $B \overset{*}{\Rightarrow} A$, merge non-terminals A and B into one non-terminal.
4. For every two non-terminals A, B such that $A \overset{*}{\Rightarrow} B$ and for every rule $B \to \alpha$, add the rule $A \to \alpha$. After this, remove all unit production rules (i.e. rules of the form $A \to B$).
5. For every terminal symbol a in the RHS of a rule of size bigger than one, replace a with a new non-terminal N_a and add the rule $N_a \to a$.

A CFG G is a *finite multiplicity grammar* if the degree of ambiguity of every string $w \in \Sigma^*$ w.r.t. G is finite. If a grammar is proper it has finite multiplicity. Two CFGs are *multiplicity equivalent* if for every $w \in \Sigma^*$, the degree of ambiguity of w is the same for both CFGs. The decidability of checking whether two CFGs are multiplicity equivalent has been an open problem for many years (Kuich and Salomaa, 1986): the problem has been proved to be decidable only for particular classes of grammars.

Results regarding decidability problems on context-free grammars can be found in many textbooks (Harrison, 1978):

1. Given two CFGs G_1 and G_2, the *equivalence* question $L(G_1) = L(G_2)$? is undecidable.
2. Given two CFGs G_1 and G_2, the *inclusion* question $L(G_1) \subseteq L(G_2)$? is undecidable.
3. Given two CFGs G_1 and G_2, the *emptiness of intersection* question $L(G_1) \cap L(G_2) = \emptyset$? is undecidable.

2.1.2 Defining Probabilistic Context-Free Grammars

A *probabilistic context-free grammar* (PCFG) is a CFG with a function assigning a probability value to each production rule and to each starting non-terminal. Formally, a PCFG is a tuple $\langle G, P \rangle$ where G is the underlying CFG $\langle N, \Sigma, R, I \rangle$ and P is a function from $(R \cup I)$ to $[0,1]$ s.t.

$$\forall A \in N, \sum_{A \to \alpha \in R} P(A \to \alpha) = 1$$

and

$$\sum_{S \in I} P(S) = 1$$

The *probability of a derivation of a non-terminal* A, denoted as $Pr(D_A)$, is the product of the probabilities of the rules used in the derivation:

$$Pr(D_A) = \prod_{A \to \alpha \in r(D_A)} P(A \to \alpha)$$

The *probability of a derivation of a PCFG*, denoted as $Pr(D_S)$, is the product of the probabilities of the rules used in the derivation and the starting non-terminal probability:

$$Pr(D_S) = P(S) \prod_{A \to \alpha \in r(D_S)} P(A \to \alpha)$$

The probability that a non-terminal A generates a string w, denoted as $Pr(A \stackrel{*}{\Rightarrow} w)$ and known as the *inside probability* of w from A, is the summation of the probabilities of all leftmost derivations of w from A:

$$Pr(A \stackrel{*}{\Rightarrow} w) = \sum_{D_A \in LD_A(w)} Pr(D_A)$$

The *probability assigned by a PCFG* G *to a string* $w \in \Sigma^*$, denoted $Pr(w)$, is the summation of the probabilities of all leftmost derivations of w from G:

$$Pr(w) = \sum_{D_S \in LD_G(w)} Pr(D_S)$$

A *weighted context-free language* (WCFL) is a pair (L, ϕ) where $L \subseteq \Sigma^*$ is a CFL and ϕ is a function from Σ^* to $\mathbb{R}$ such that $w \in L \Leftrightarrow \phi(w) > 0$. For any PCFG G, The CFL L generated by the underlying CFG of G and $\phi(w) = Pr(w)$ form the WCFL (L, ϕ) of G. A *probabilistic context-free language* (PCFL) is a WCFL (L, ϕ) generated from a PCFG such that

$$\sum_{w \in L} \phi(w) = 1$$

Interestingly, not every PCFL can be generated by a PCFG and not every WCFL of a PCFG is a PCFL. For example, take the well known CFL $\{a^n b^n | n \geq 0\}$ as the support for a PCFL with the probability function

$$\phi(a^n b^n) = \frac{1}{e.n!}.$$

Booth and Thompson (1973) show that there exists no PCFG capable of generating this PCFL. The intuition behind this is that the probability function of a PCFG can only grow polynomially in the length of the string, whilst this function grows exponentially (because of $n!$) in the length of the string (Wetherell, 1980). On the other hand, consider the following PCFG:

$$S \to SS \ \tfrac{4}{5} \quad S \to a \ \tfrac{1}{5}$$

The WCFL of this PCFG is not a PCFL because the probabilities of all strings in the language add up to less than one. We can calculate the total probability by computing the infinite sum of probabilities of string a, aa, aaa, ... as follows:

$$\begin{aligned}\text{Total Probability} &= \sum_{k=0}^{\infty} C_k \left(\frac{4}{5}\right)^k \left(\frac{1}{5}\right)^{k+1} \\ &= \frac{1}{5} \cdot \sum_{k=0}^{\infty} \left(\frac{1}{k+1}\right) \binom{2k}{k} \left(\frac{4}{25}\right)^k \\ &= \frac{1}{5} \cdot \frac{1-\sqrt{1-4\left(\frac{4}{25}\right)}}{2\left(\frac{4}{25}\right)} \\ &= 0.25\end{aligned}$$

where C_k is the k^{th} Catalan number. This is because the different binary trees that generate a^{k+1}, for $k \geq 0$, are in fact all the possible binary trees with $k+1$ leaves.

Unless otherwise specified, when we refer to PCFGs we will assume that they generate PCFLs. In Section 2.2, we give more details on how to check whether a PCFG actually generates a PCFL.

We denote by $\mathbb{L}(G)$ the support language of G, i.e. the set of strings of non null probability. There exists an effective procedure which, given a proper CFG G, builds a PCFG G' such that $\forall x \in \Sigma^*, Pr_{G'}(x) > 0 \iff x \in \mathbb{L}(G)$. We call this procedure MP for Make Probabilistic. One possible procedure for MP is to first assign uniform probabilities to the given CFG, thus obtaining a possibly inconsistent PCFG which then can be converted into a consistent PCFG using the procedure explained in (Gecse and Kovács, 2010). This procedure changes the probabilities of a given weighted grammar until it becomes consistent (i.e. a PCFG), whilst still generating the same non-probabilistic language.

The equivalence problem for PCFGs can be defined as follows:

Definition 2.1.1. *Two PCFGs G_1 and G_2 are **equivalent** if $\forall x \in \Sigma^*, Pr_{G_1}(x) = Pr_{G_2}(x)$. We denote by $\langle EQ, \mathbf{PCFG}(\Sigma)\rangle$ the decision problem: are two PCFGs G_1 and G_2 equivalent?*

The following result holds for probabilistic pushdown automata, which are shown in (Abney et al., 1999) to be equivalent to PCFGs.

Proposition 2.1.1. *(Forejt et al., 2014) The $\langle EQ, \mathbf{PCFG}(\Sigma)\rangle$ problem is interreducible with the multiplicity equivalence problem for* CFGs.

Note that any general λ-free PCFG can be transformed (in polynomial time) into an equivalent PCFG in CNF (Abney et al., 1999). This can be done by using the same procedure to transform CFGs to CNF. The probabilities on the added rules can always be distributed so that the obtained grammar remains probabilistically equivalent.

2.2 Consistency

In Section 2.1.2, we showed that not every PCFG generates a PCFL. We say that a PCFG is *consistent* if it generates a PCFL. By definition, consistency is a necessary condition for a PCFG to describe a probability distribution. Therefore, it is important to know whether a PCFG is consistent or not.

Wetherell (1980) shows that a PCFG is consistent if the spectral radius of its stochastic expectation matrix (which we define below) is strictly less than 1. For the case when the spectral radius is exactly equal to

1, Wetherell (1980) states that more complicated tests are needed to check for consistency. More recently, Etessami and Yannakakis (2009b) showed that in case a PCFG is proper, if the spectral radius is exactly equal to 1 than the PCFG is consistent.

The stochastic expectation matrix of a PCFG G is built by multiplying two square matrices $\mathbf{Q}$ and $\mathbf{C}$ which can easily be built from G.

Matrix $\mathbf{Q}$ has $|N|$ rows and $|R|$ columns indexed by non-terminals and production rules respectively. The value in element $[A, X \to \alpha]$ is $P(X \to \alpha)$ if non-terminal A is equal to non-terminal X and 0 otherwise.

Matrix $\mathbf{C}$ has $|R|$ rows and $|N|$ columns indexed by production rules and non-terminals respectively. The value in element $[X \to \alpha, A]$ is the number of occurrences of non-terminal A on the RHS of production rule $X \to \alpha$.

The stochastic expectation matrix $\mathbf{M}$ of a PCFG, is a square matrix with $|N|$ rows and columns indexed by non-terminals. $\mathbf{M}$ is simply $\mathbf{Q} \cdot \mathbf{C}$. The value in element $[A, B]$ is the expected number of times B will occur when A is rewritten using exactly one production rule.

The spectral radius of a square matrix $\mathbf{M}$ is the modulus of the largest eigenvalue of $\mathbf{M}$. Wetherell (1980) gives a method for calculating the spectral radius using Gerschgorin's algorithm (Stewart, 1973) whilst Etessami and Yannakakis (2009b) use Jacobian matrix forms alongside graph theoretic techniques. Gecse and Kovács (2010) give an overview of these and other methods used.

2.3 Parsing and Probability of a String

An algorithm for finding the probability that a PCFG assigns to a string is crucial to have. Without this algorithm, we do not have access to the distribution defined by the PCFG over the language generated by the underlying CFG. Without access to the distribution, a PCFG is practically no better than its underlying CFG.

In section 2.1.2, we defined the probability assigned by a PCFG to a string as the summation of the probabilities of all leftmost derivations. This definition does not directly translate into an efficient procedure. First of all, it does not describe how derivations can be found and secondly, the number of leftmost derivations can be exponential (or even infinite) in the length of the string.

Fortunately, efficient procedures (that use dynamic programming) exist to solve this problem. These procedures were initially designed to find and efficiently store all the left-most derivations of a given string w from a given grammar G; in other words, they were designed as parsers for CFGs. These procedures can be easily extended to work with PCFGs by also returning the probability of the given string. In this section, we describe and analyse two of such procedures, the Cocke–Younger–Kasami (CYK) algorithm and the Earley algorithm, and show how they can be used to find the probability of a string.

2.3.1 CYK Algorithm

The CYK algorithm assumes that the grammar to be parsed is in CNF. As noted in section 2.1, every CFG (resp. PCFG) can be effectively transformed into an equivalent CFG (resp. PCFG) in CNF. So we do not lose generality for the purposes of checking whether a string is in the language of a CFG or for finding the probability of a string.

The CYK algorithm decides whether a given CFG G generates a given string $w = a_0 a_1 \dots a_{n-1}$ (note that we use zero-based numbering to index strings). It works by filling (part of) a three dimensional boolean array T of size $n \times n \times |N|$, $|N|$ being the number of non-terminals. We shall index the first two dimensions

numerically (using zero-based numbering) and the third dimension using the non-terminals. Each element $[i, j, A]$ in T holds the truth value of the following statement: "Non-terminal A generates $a_i \dots a_{i+j}$". Clearly, if for any starting non-terminal S, $T[0, n-1, S]$ is true, then we can say that G generates w, otherwise not.

All values in the table are initialized as false. The algorithm fills the table recursively. The base case is on the elements which span over only one symbol (i.e. the index of the second dimension is 0). These can be filled as follows:

$$\bigvee_{i \in 0..n-1} \bigvee_{A \to a_i \in R} T[i, 0, A] = True$$

This is because for every symbol a, only a rule of the form $A \to a$ can generate a. The recursive case is built on the following condition: if B generates u and C generates v and there is a rule $A \to BC$, then A must generate uv. So, elements can be incrementally added as follows:

$$\bigvee_{j \in 1..n-1} \bigvee_{\substack{i \in 0..n-j-1 \\ k \in 0..j-1}} \bigvee_{A \to BC \in R} if\ T[i,k,B]\ and\ T[i+k+1, j-k-1, C]\ then\ T[i,j,A] = True$$

The general recursive equation is as follows:

$$A \overset{*}{\Rightarrow} a_i \dots a_{i+j}? = \begin{cases} True & j == 0\ and\ A \to a_i \in P \\ True & A \to BC \in P\ and \bigexists_{\substack{A \to BC \in P \\ k \in 0..j-1}} B \overset{*}{\Rightarrow} a_i \dots a_{i+k}\ and\ C \overset{*}{\Rightarrow} a_{i+k+1} \dots a_{i+j} \\ False & otherwise \end{cases}$$

Probability of a string

We can extend this algorithm to compute the probability assigned by a PCFG G to a string $w = a_0 a_1 \dots a_{n-1}$. Instead of a boolean array, an array T of probability values $[0, 1]$ is used (all values are initialized to zero). This time, each element $[i, j, A]$ in T holds the probability that non-terminal A generates $a_i \dots a_{i+j}$. This value is denoted as $Pr(A \overset{*}{\Rightarrow} a_i \dots a_{i+j})$ or $I(i, j, A)$. So, the probability of w w.r.t. G is exactly the summation of $P(S).T[0, n-1, S]$ for all starting non-terminals S, where $P(S)$ is the probability that S is used as the starting non-terminal.

The table is filled in a similar manner. For the base case:

$$\bigvee_{i \in 0..n-1} \bigvee_{A \to a_i \in R} T[i, 0, A] = P(A \to a_i)$$

This is because the probability that A generates a symbol a is exactly the probability of the rule $A \to a$. The recursive case is built on the following two conditions:

1. $Pr(A \overset{*}{\Rightarrow} a_i \dots a_{i+j})$ is equal to the sum for all $A \to \alpha$ (i.e. all rules with A on the LHS) of the probability that A generates $a_i \dots a_{i+j}$ using the rule $A \to \alpha$
2. The probability of A generating $a_i \dots a_{i+j}$ using the rule $A \to BC$ is the sum over all $k \in 0..j-1$ (i.e. sum over all 'cuts' of $a_i \dots a_{i+j}$) of $P(A \to BC).Pr(B \overset{*}{\Rightarrow} a_i \dots a_{i+k}).Pr(C \overset{*}{\Rightarrow} a_{i+k+1} \dots a_{i+j})$

Using these two conditions, the table is filled as follows:

$$\bigvee_{j \in 1..n-1} \bigvee_{\substack{i \in 0..n-j-1 \\ k \in 0..j-1}} \bigvee_{A \to BC \in R} T[i,j,A] = T[i,j,A] + P(A \to BC)\,.\,T[i,k,B]\,.\,T[i+k+1, j-k-1, C]$$

The general recursive equation is as follows:

$$I(i,j,A) = \begin{cases} P(A \to a_i) & j == 0 \text{ and } A \to a_i \in P \\ \sum\limits_{\substack{A \to BC \in P \\ k \in 0..j-1}} P(A \to BC) \,.\, I(i,k,B) \,.\, I(i+k+1, j-k-1) & j > 0 \\ 0 & otherwise \end{cases}$$

The most probable parse tree

This algorithm can again be extended in order to find the most probable parse tree for $w = a_0 a_1 \dots a_{n-1}$. This time, each element $[i,j,A]$ in T holds the probability of the most probable parse tree of A deriving $a_i \dots a_{i+j}$. We denote this value as $Pr_{max}(A \stackrel{*}{\Rightarrow} a_1 \dots a_{i+j})$ or $I_{max}(i,j,A)$. The base case remains the same, since only one parse tree is possible for a single symbol. The recursive case is built on the following two conditions:

1. $Pr_{max}(A \stackrel{*}{\Rightarrow} a_1 \dots a_{i+j})$ is equal to the maximum probability for all $A \to \alpha$ of the most probable parse tree of A generating $a_1 \dots a_{i+j}$ using the rule $A \to \alpha$.
2. The probability of the most probable parse tree of A generating $a_1 \dots a_{i+j}$ using the rule $A \to BC$ is the maximum probability over $k \in 0..j-1$ (i.e. maximum probability over all 'cuts' of $a_i \dots a_{i+j}$) of $P(A \to BC).Pr_{max}(B \stackrel{*}{\Rightarrow} a_i \dots a_{i+k}).Pr_{max}(C \stackrel{*}{\Rightarrow} a_{i+k+1} \dots a_{j-k-1})$

Using these two conditions, the table is filled as follows:

$$\bigvee_{j \in 1..n-1} \; \bigvee_{\substack{i \in 0..n-j-1 \\ k \in 0..j-1}} \; \bigvee_{A \to BC \in R} T[i,j,A] = max(T[i,j,A] \;,\; P(A \to BC) \,.\, T[i,k,B] \,.\, T[i+k+1, j-k-1, C])$$

The probability of the most probable parse tree for w is equal to the highest probability of $P(S).T[0,n-1,S]$ for all $S \in I$. Finding the most probable parse tree for w can be done by simply keeping track of the rules which yield the maximum probabilities. Note that finding the most probable parse tree is of interest in applications where a PCFG is used to assign structure (in the form of one parse tree or a limited number of parse trees) to a given string.

The general recursive equation is as follows:

$$I_{max}(i,j,A) = \begin{cases} P(A \to a_i) & j == 0 \text{ and } A \to a_i \in P \\ \max\limits_{\substack{A \to BC \in P \\ k \in 0..j-1}} P(A \to BC) \,.\, I_{max}(i,k,B) \,.\, I_{max}(i+k+1, j-k-1) & j > 0 \\ 0 & otherwise \end{cases}$$

Complexity

The CYK algorithm and its extensions take $\mathcal{O}(|P| \cdot n^3)$ time, where $|P|$ is the number of production rules of the grammar to be parsed and n is the length of the string to be parsed. The n^3 term stands for the fact that every split of every substring is traversed, whilst the $|P|$ term is there because all production rules are traversed to find matching rules for each split. The space complexity is $\mathcal{O}(n^2 \cdot |N|)$, which is the size of the CYK table.

2.3.2 Earley Algorithm

Unlike the CYK algorithm, the Earley algorithm is considered as a top-down rather than a bottom-up parsing algorithm. This is because it works its way down from the starting production rules to the symbols in the string rather than the other way round. Also, unlike the CYK algorithm, it does not assume that the grammar to be parsed is in CNF.

Although the Earley algorithm can be modified to work with unrestricted CFGs, the standard Earley algorithm works with particular forms of grammars used to model natural language syntax. These grammars satisfy the following constraints:

– Every production rule is of the form $N \to N^+$ or $N \to \Sigma$

– The set of non-terminals can be split into two disjoint sets, parts of speech (POS) non-terminals and phrasal non-terminals. The LHS non-terminals of rules of the form $N \to \Sigma$ are POS non-terminals while the rest are phrasal non-terminals.

Note that any CFG (resp. PCFG) can be transformed into an equivalent CFG (resp. PCFG) which satisfies these constraints. This can easily be done by substituting terminals a in RHS of rules of length bigger than 1 with newly added non-terminal N_a and adding POS rules $N_a \to a$. Probabilities can easily be assigned in such a way that the resulting grammar remains probabilistically equivalent.

The Earley algorithm works by traversing the given string $w = a_0 \dots a_{n-1}$ once from left to right and adding states to a chart as it goes along. States show how much a particular production rule has been processed so far and which substring it generates at that point. The presence of some particular states in the last chart entry indicates that the string is recognized by the grammar. We will first explain the syntax and semantics of a state, followed by an explanation of what the chart is, and finally we show how the chart is filled with states and which states determine whether a string is recognized or not by the grammar.

The states and the chart

A state takes one of the following two forms:

$$[A \to \alpha \bullet B\beta\,,\, i\,,\, j]$$
$$[A \to a_i \bullet\,,\, i\,,\, i+1]$$

where:

– $A \to \alpha B\beta$ and $A \to a_i$ are production rules (where $\alpha, \beta \in N^*$)

– $A \to \alpha \bullet B\beta$ and $A \to a_i \bullet$ are known as dotted rules. The $\bullet$ symbol for the first rule can also be at the end of the rule (i.e. $A \to \alpha\bullet$, where $\alpha \in N^+$)

– i and j are two indices from $[0, n-1]$ and $[0, n]$ respectively such that $i \leq j$

A state $[A \to \alpha \bullet B\beta\,,\, i\,,\, j]$ indicates two things:

1. Production rule $A \to \alpha B\beta$ has been processed up until the last non-terminal of α. If α is empty, then the production rule has not yet been processed and if $\bullet$ is at the end of the rule then the production rule has been fully processed.
2. $\alpha \overset{*}{\Rightarrow} a_i \dots a_{j-1}$. In case α is empty, then the production rule does not generate anything (since it has not yet been processed) and thus i is equal to j.

The chart C is an array with $n+1$ entries (from 0 to n). Each entry holds an ordered set of states. $C[j]$ holds states with index j (i.e. states whose production rules generate substrings up to a_{j-1}). $C[0]$ contains states with unprocessed production rules starting from the beginning of the string and $C[n]$ contains fully processed production rules generating suffixes of w. Having a state $[S \to \alpha\bullet\,,\, 0\,,\, n]$ in $C[n]$ (for any $S \in I$) means that the grammar generates w.

Filling the chart

The chart is incrementally filled in n steps, starting from $C[0]$ up to $C[n]$. A state with the second index j will be in $C[j]$. States are never removed from the chart. Three operators are used to fill the chart: the PREDICTOR, SCANNER and COMPLETER. Each operator takes one state as input and constructs new states from it. The COMPLETER needs access to states in previous chart entries, whilst the PREDICTOR and SCANNER do not need this. At the j^{th} step, the PREDICTOR and COMPLETER add states to $C[j]$ whilst the SCANNER adds states to $C[j+1]$. The three operators work as follows:

– Given a state $[A \rightarrow \alpha \bullet B\beta, i, j]$ where B is a phrasal non-terminal, the PREDICTOR adds the states $[B \rightarrow \bullet\gamma, j, j]$ in $C[j]$ for every production rule $B \rightarrow \gamma$ with B on the LHS. Therefore, the PREDICTOR adds states with all the possible rules which the non-terminal just after the $\bullet$ can use. Note that the rules in the added states are unprocessed since the indices are the same.

– Given a state $[A \rightarrow \alpha \bullet B\beta, i, j]$ where B is a POS non-terminal, the SCANNER adds the state $[B \rightarrow a_j \bullet, j, j+1]$ in $C[j+1]$ if there exists a production rule $B \rightarrow a_j$.

– Given a state $[B \rightarrow \alpha\bullet, k, j]$ (i.e. a state with a fully processed production rule), the COMPLETER searches for states in $C[k]$ of the form

$$[A \rightarrow \alpha \bullet B\beta, i, k]$$

and for each such state, the state:

$$[A \rightarrow \alpha B \bullet \beta, i, j]$$

is added to $C[j]$. In other words, the COMPLETER adds states with the $\bullet$ symbol moved one step to the right (i.e. from behind B to after B) when it finds a fully processed rule for B with matching indices.

Algorithm 1 shows how and when these three operators are used to fill the whole chart. Note that the initial states in $C[0]$ have unprocessed rules with starting non-terminals on the LHS. Also note that each time a state is added in a chart entry, it is placed at the end of the ordered set. Therefore, when iterating on the same set where states are added, the newly added states will always be reached by the iteration. After filling the chart, we know that w is accepted by the grammar if there exists a state $[S \rightarrow \alpha\bullet, 0, n]$ in $C[n]$ (for any $S \in I$), otherwise w is not accepted by the grammar.

The probability of the string can be calculated by multiplying the probabilities associated with successive states in the Earley chart (see (Stolcke, 1995) for more details).

Algorithm 1: Earley Algorithm

Input: A CFG G; a string $w = a_0 \dots a_{n-1}$
Output: *True* if $w \in L(G)$, *False* otherwise

$C[0] \leftarrow \{[S \rightarrow \bullet\, \alpha\,, 0\,, 0] \mid S \in I, S \rightarrow \alpha\}$;
for *i = 0 to n* **do**
  foreach *state in C[i]* **do**
    if production rule in *state* is fully processed **then** COMPLETER(state) ;
    else if the non-terminal after $\bullet$ is a POS non-terminal **then** SCANNER(state) ;
    else PREDICTOR(state) ;
  end
end
if $\{[S \rightarrow \alpha \bullet\,, 0\,, n] \mid S \in I, S \rightarrow \alpha\} \cap C[n] = \emptyset$ **then return** *False* ;
else return *True* ;

procedure COMPLETER($[B \rightarrow \alpha\bullet, k\,, j]$)
  foreach $[A \rightarrow \alpha \bullet B\beta\,, i\,, k]$ **in** $C[k]$ **do**
    Add $[A \rightarrow \alpha B \bullet \beta\,, i\,, j]$ to $C[j]$;
  end

procedure SCANNER($[A \rightarrow \alpha \bullet B\beta\,, i\,, j]$)
  if $B \rightarrow a_j \in R$ **then**
    Add $[B \rightarrow a_j\bullet\,, j\,, j+1]$ to $C[j+1]$;
  end

procedure PREDICTOR($[A \rightarrow \alpha \bullet B\beta\,, i\,, j]$)
  foreach $B \rightarrow \gamma \in R$ **do**
    Add $[B \rightarrow \bullet\,\gamma\,, j\,, j]$ to $C[j]$;
  end

Complexity

The worst case running time complexity of the Earley algorithm is $\mathcal{O}(|G|^2 n^3)$ (An improved Earley algorithm is given in (Graham et al., 1980) with a worse case running time of $\mathcal{O}(|G|n^3)$). However, for certain types of grammars, the Earley algorithm has a lower worst time complexity. For example, the complexity is $\mathcal{O}(|G|^2 n^2)$ for grammars with bounded ambiguity and $\mathcal{O}(|G|^2 n)$ for deterministic grammars (Stolcke, 1995). The space complexity in terms of the length of the string is $\mathcal{O}(n^2)$.

In practice, in scenarios when the grammar is not big and not highly ambiguous and/or the string is relatively long (> 30), then the Earley algorithm will most probably be more efficient than the CYK algorithm. The CYK algorithm might perform better on relatively big or highly ambiguous grammars and when the string to be parsed is short.

2.4 Prefix probability

A more complicated problem than finding the probability of a string is to find the probability of a prefix. Formally, given a PCFG G and a string u, the problem is to compute:

$$Pr(u\Sigma^*) = \sum_{v \in \Sigma^*} Pr(uv)$$

Being able to compute this allows us to compute the probability of a symbol following a given prefix u. This can be done as follows:

$$Pr(\text{next symbol is } a \mid \text{prefix } u) = \frac{Pr(ua\Sigma^*)}{Pr(u\Sigma^*)}$$

This will prove useful in evaluating how good a PCFG is as a language model. In this section, we explain the standard algorithm for computing the probability of a prefix string: the LRI algorithm (Jelinek and Lafferty, 1991).

2.4.1 Prerequisites

In order to show how prefix probabilities are computed, we first have to show how the following three probabilities are computed:

– $Pr_L(A \stackrel{*}{\Rightarrow} X)$: the probability that non-terminal A leftmost reaches non-terminal X. This is calculated as follows:

$$Pr_L(A \stackrel{*}{\Rightarrow} X) = \sum_{v \in \Sigma^*} Pr(A \stackrel{*}{\Rightarrow} Xv)$$

– $Pr_L(A \stackrel{*}{\Rightarrow} X \to YZ)$: the probability that non-terminal A leftmost reaches the production rule $X \to YZ$. This is simply $Pr_L(A \stackrel{*}{\Rightarrow} X).P(X \to YZ)$.

– $Pr(A \stackrel{*}{\Rightarrow} a\ldots)$: this is the probability that non-terminal A generates the single symbol a as a prefix. In terms of $Pr_L(A \stackrel{*}{\Rightarrow} X)$, this is calculated as follows:

$$Pr(A \stackrel{*}{\Rightarrow} a\ldots) = P(A \to a) + \sum_{X \in N} Pr_L(A \stackrel{*}{\Rightarrow} X).P(X \to a)$$

Computing $Pr_L(A \stackrel{*}{\Rightarrow} X \to YZ)$ and $Pr(A \stackrel{*}{\Rightarrow} a\ldots)$ is trivial if we have a way to compute $Pr_L(A \stackrel{*}{\Rightarrow} X)$. What follows is an explanation of how $Pr_L(A \stackrel{*}{\Rightarrow} X)$ can be computed.

Let $\mathbf{P}$ be a square matrices of size $|N| \times |N|$ and indexed by non-terminals such that:

$$\mathbf{P}[A,X] = \sum_{B \in N} P(A \to XB)$$

So, $\mathbf{P}[A,X]$ stores the probability that non-terminal A leftmost reaches non-terminal X in one step. We can easily build $\mathbf{P}$.

Let $\mathbf{Q}$ be a square matrices of size $|N| \times |N|$ such that:

$$\mathbf{Q} = \mathbf{P} + \mathbf{P}^2 + \mathbf{P}^3 + \ldots + \mathbf{P}^k + \ldots$$

where $\mathbf{P}^k$ denotes the k-fold multiplication of the matrix $\mathbf{P}$ by itself. So, $\mathbf{Q}[A,X]$ stores the probability that non-terminal A leftmost reaches non-terminal X in one or more steps, which is exactly $Pr_L(A \stackrel{*}{\Rightarrow} X)$ (i.e. the value we want to compute). We can change the equation for $\mathbf{Q}$ as follows:

$$\begin{aligned}
\mathbf{Q} &= \mathbf{P} + \mathbf{P}^2 + \mathbf{P}^3 + \ldots + \mathbf{P}^k + \ldots \\
\mathbf{Q}.\mathbf{P} &= \mathbf{P}^2 + \mathbf{P}^3 + \ldots + \mathbf{P}^k + \ldots \\
\mathbf{Q} - \mathbf{Q}.\mathbf{P} &= \mathbf{P} \\
\mathbf{Q} &= \mathbf{P}[\mathbf{I} - \mathbf{P}]^{-1}
\end{aligned}$$

where $\mathbf{I}$ is the identity matrix of the same size as $\mathbf{P}$ and $[\mathbf{I}-\mathbf{P}]^{-1}$ is the inverse matrix of $\mathbf{I}-\mathbf{P}$. With this equation, it is easy to build $\mathbf{Q}$, and thus we have a way to compute $Pr_L(A \stackrel{*}{\Rightarrow} X)$ which in turn allows us to compute $Pr_L(A \stackrel{*}{\Rightarrow} X \to YZ)$ and $Pr(A \stackrel{*}{\Rightarrow} a\ldots)$.

2.4.2 LRI Algoritm

The computation of prefix probabilities relies on the following proposition, which holds for any CNF grammar.

Proposition 2.4.1. *If non-terminal A generates $a_i \ldots a_{i+m}$ and generates $a_i \ldots a_{i+j}$ as a prefix of length bigger than 1 (i.e. $1 \leq j \leq m$) then for every parse tree of A that generates $a_i \ldots a_{i+j}$ as a prefix, there exists a rule $X \to YZ$ and a $k \in 0..j-1$ such that*

- *Condition 1: A leftmost reaches X.*
- *Condition 2: Y generates $a_i \ldots a_{i+k}$.*
- *Condition 3: Z generates $a_{i+k+1} \ldots a_{i+j}$ as a prefix.*

Proof. By induction on m.

Base case, $m = 1$
The base case is when A generates a string of length 2. This can only be done using one rule $A \to YZ$ and two terminal rules $Y \to a_i$ and $Z \to a_{i+1}$. It is easy to see that all conditions hold in this case.

Inductive case
Let $A \to BC$ be the first production used by A to generate $a_i \ldots a_{i+m}$. There are two scenarios:

1. B generates part of the prefix $a_i \ldots a_{i+j}$ (up to a_{i+k} for some $k \in 0...j-1$) which makes Condition 2 true. Condition 3 is also true because C generates $a_{i+k+1} \ldots a_{i+m}$, which contains the prefix $a_{i+k+1} \ldots a_{i+j}$. Condition 1 is trivially true.
2. B generates at least the whole prefix $a_i \ldots a_{i+j}$ (up to a_{i+k} for some $k \in j...m-1$). In this case, we can ignore the suffix $a_{i+k+1} \ldots a_{i+j}$ and consider the same problem but with non-terminal B instead of non-terminal A and $a_i \ldots a_{i+k}$ instead of $a_i \ldots a_{i+m}$. By the inductive hypothesis, all the conditions for this new problem are true. So, since trivially A leftmost reaches $A \to BC$, then the original problem is also true.

□

Therefore, the probability that non-terminal A generates prefix $a_i \ldots a_{i+j}$ can be calculated recursively as follows:

$$Pr(A \stackrel{*}{\Rightarrow} a_i \ldots a_{i+j} \ldots) = \begin{cases} Pr(A \stackrel{*}{\Rightarrow} a_i \ldots) & j == 0 \\ \sum_{X \to YZ} Pr_L(A \stackrel{*}{\Rightarrow} X \to YZ) \left[\sum_{k \in 0..j-1} I(i,k,Y).Pr(Z \stackrel{*}{\Rightarrow} a_{i+k+1} \ldots a_{i+j} \ldots) \right] & otherwise \end{cases}$$

Note that $Pr_L(A \stackrel{*}{\Rightarrow} X \to YZ)$ is for condition 1 in proposition 2.4.1, $I(i,k,Y)$ for condition 2 and $Pr(Z \stackrel{*}{\Rightarrow} a_{i+k+1} \ldots a_{i+j} \ldots)$ for condition 3. We know how to compute $Pr(A \stackrel{*}{\Rightarrow} a_i \ldots)$ and $Pr_L(A \stackrel{*}{\Rightarrow} X \to YZ)$ from section 2.4.1 and $I(i,k,Y)$ from section 2.3.1. $Pr(Z \stackrel{*}{\Rightarrow} a_{i+k+1} \ldots a_{i+j} \ldots)$ is computed recursively.

Finally, the probability that a grammar generates prefix $a_i \ldots a_{i+j}$ is simply:

$$Pr(a_i \ldots a_{i+j} \ldots) = \sum_{S \in I} P(S) \,.\, Pr(S \stackrel{*}{\Rightarrow} a_i \ldots a_{i+j} \ldots)$$

2.5 Parameter Estimation

An important problem is that of estimating the unknown probabilities of a PCFG G given a sample of strings T generated from G. This is normally preceded by a structure discovery step, in which the underlying CFG of G is induced from T (this is in fact our main contribution in this thesis). Whilst structure discovery has proved to be a very difficult task and few positive results have been obtained, parameter estimation has been more successful and a well established algorithm has been developed to tackle this task: the inside-outside algorithm. In this section, we explain and discuss this algorithm.

2.5.1 Setting

Suppose that we have access to a sample T of strings with their parse trees which were generated from a PCFG G. Suppose that we only know the underlying structure of G (i.e. the probabilities are unknown), and our task is to estimate the probabilities of G using T and the parse trees. If we follow the maximum likelihood principle, which states that the best parameters of a model are those which make the observed data most likely, then we should estimate the probabilities of G using the relative frequencies as follows:

$$P(A \to \alpha) = \frac{Count(A \to \alpha)}{\sum_{A \to \beta \in R} Count(A \to \beta)} \tag{2.1}$$

$Count(A \to \alpha)$ is the number of times rule $A \to \alpha$ is used in all the parse trees of the sample, and the denominator is the number of times non-terminal A is used in all the parse trees of the sample. However, if parse trees are not available, we cannot compute these counts. The inside-outside algorithm solves this problem by using expected counts (rather than 'real' counts) which can be computed from T alone. For expected counts to be computed, the inside and outside probabilities need to be computed. The inside probability $I(i,j,A)$ is simply $Pr(A \stackrel{*}{\Rightarrow} w_i \dots w_{i+j})$ and we showed how to compute this using the CYK algorithm in section 2.3.1. We now show how to compute the outside probabilities, after which we explain how expected counts are computed. We can then explain the inside-outside algorithm as a whole, which simply iteratively computes the expected counts and stops whenever the likelihood of the sample converges.

2.5.2 The outside probability

Given a string $w = a_0 \dots a_{n-1}$ generated by a PCFG G, the outside probability $O(i,j,A)$ is the probability that a starting non-terminal of G generates $a_0 \dots a_{i-1} A a_{i+j+1} \dots a_{n-1}$ such that $A \stackrel{*}{\Rightarrow} a_i \dots a_{i+j}$. This value can be computed recursively in a similar manner as is done in the CYK algorithm, but using top-down instead of bottom-up recursion.

The base case relies on the following two simple facts:

1. For every starting non-terminal S, if $S \stackrel{*}{\Rightarrow} w$ then $O(0,n-1,S) = P(S)$, otherwise $O(0,n-1,S) = 0$. This is because trivially $S \stackrel{*}{\Rightarrow} S$ with probability 1 and the only probability used is to select S as the starting non-terminal.
2. For every non-starting non-terminal A, $O(0,n-1,A) = 0$, because otherwise A would be a starting non-terminal.

Therefore, the equation for the base case is as follows (note that we can decide if $A \stackrel{*}{\Rightarrow} w$ using the CYK or Earley algorithm):

$$O(0,n-1,A) = \begin{cases} P(A) & A \in I \text{ and } A \stackrel{*}{\Rightarrow} w \\ 0 & \text{otherwise} \end{cases}$$

The recursive case relies on the following statements:

1. If $S \stackrel{*}{\Rightarrow} a_0 \dots a_{i-1} A a_{i+j+1} \dots a_{n-1}$, where either $i > 0$ or $i + j < n - 1$, then there must exist a non-terminal X (X can be equal to S) such that S reaches X and A is reached in one step from X using $X \to YA$ or $X \to AY$ (or both).
2. Following from statement 1, $O(i, j, A)$ for the case when A is reached through $X \to YA$ is the summation for $k \in 1..i$ of $O(i-k, j+k, X).P(X \to YA).I(i-k, k-1)$, where:

$$\begin{aligned} S &\stackrel{*}{\Rightarrow} a_0 \dots a_{i-k-1} X\, a_{i+j+1} \dots a_{n-1} \\ &\Rightarrow a_0 \dots a_{i-k-1} YA\, a_{i+j+1} \dots a_{n-1} \\ &\stackrel{*}{\Rightarrow} a_0 \dots a_{i-1} A\, a_{i+j+1} \dots a_{n-1} \end{aligned}$$

3. Similarly, $O(i, j, A)$ for the case when A is reached through $X \to AY$ is the summation for $k \in 1..n - 1 - i - j$ of $O(i, j+k, X).P(X \to AY).I(i+j+1, k-1)$, where:

$$\begin{aligned} S &\stackrel{*}{\Rightarrow} a_0 \dots a_{i-1} X\, a_{i+j+k+1} \dots a_{n-1} \\ &\Rightarrow a_0 \dots a_{i-1} AY\, a_{i+j+k+1} \dots a_{n-1} \\ &\stackrel{*}{\Rightarrow} a_0 \dots a_{i-1} A\, a_{i+j+1} \dots a_{n-1} \end{aligned}$$

4. Therefore, $O(i, j, A)$ is the summation of statement 2 over all the possible $X \to YA$ rules, added with the summation of statement 3 over all the possible $X \to AY$ rules.

Therefore, the equation for the recursive case is as follows:

$$\begin{aligned} O(i, j, A) = &\sum_{X \to AY \in R} \sum_{k \in 1..i} O(i-k, j+k, X).P(X \to YA).I(i-k, k-1) + \\ &\sum_{X \to AY \in R} \sum_{k \in 1..n-1-i-j} O(i, j+k, X).P(X \to AY).I(i+j+1, k-1) \end{aligned}$$

2.5.3 The expected counts

Given a PCFG G (with probabilities) and a string $w = a_0 \dots a_{n-1}$ by G, we are interested in computing the expected number of times a production rule $A \to \alpha$ appears in parse trees of w, denoted as $E_w[A \to \alpha]$. This is calculated as follows:

$$E_w[A \to \alpha] = \frac{\displaystyle\sum_{i \in 0..n-1} \sum_{j \in 0..n-i-1} Pr \left(\begin{array}{l} S \stackrel{*}{\Rightarrow} a_0 \dots a_{i-1} A a_{i+j+1} \dots a_{n-1} \\ \quad \Rightarrow a_0 \dots a_{i-1} \alpha a_{i+j+1} \dots a_{n-1} \\ \quad \stackrel{*}{\Rightarrow} w \end{array} \right)}{Pr(w)}$$

The numerator is simply the summation of the probabilities of all possible ways the rule $A \to \alpha$ can be used in the generation of w. If we consider a rule of the form $A \to a$, then the summation is on $O(i, i, A).P(A \to a)$ and thus:

$$E[A \to a] = \frac{P(A \to a)}{Pr(w)} \sum_{i \in 0..n-1} O(i, i, A) \tag{2.2}$$

For rules of the form $A \to BC$, the summation is on $O(i, j, A).P(A \to BC).I(i, k, B).I(i+k+1, j-k-1, C)$ and therefore:

$$E[A \to BC] = \frac{P(A \to BC)}{Pr(w)} \sum_{j \in 1..n-1} \sum_{\substack{i \in 0..n-j-1 \\ k \in 0..j-1}} O(i,j,A) I(i,k,B) I(i+k+1, j-k-1, C) \tag{2.3}$$

2.5.4 The Inside Outside Algorithm

The algorithm starts by assigning probabilities to the given CFG, such that the grammar obtained is a consistent PCFG. These probabilities can be assigned arbitrarily or following some criterion based on the sample. Following the maximum likelihood principle, the algorithm tries to maximize the log-likelihood of the sample by iteratively computing better estimations of the grammar's probabilities. The log-likelihood of the sample is computed as follows:

$$LL(G,T) = \sum_{w \in T} log(Pr(w))$$

The probabilities of the rules are estimated using equation 2.1, where the expected counts (using equations 2.2 and 2.3) are used instead of the real counts. Algorithm 2 describes the whole process.

Algorithm 2: Inside-Outside Algorithm

Input: A CFG; a sample of strings T; convergence parameter ε
Output: A PCFG G

Choose initial probabilities for the given CFG, to obtain a PCFG $G = \langle N, \Sigma, R, I, P \rangle$;
$prev\_likelihood \leftarrow LL(G,T)$;
while *True* **do**
  foreach $A \to \alpha \in R$ **do**
    $Count(A \to \alpha) \leftarrow 0$;
    foreach $w \in T$ **do**
      $Count(A \to \alpha) \leftarrow Count(A \to \alpha) + E_w[A \to \alpha]$;
    end
  end
  foreach $A \to \alpha \in R$ **do**
    $P(A \to \alpha) \leftarrow \frac{Count(A \to \alpha)}{\sum_{A \to \beta \in R} Count(A \to \beta)}$;
  end
  $likelihood \leftarrow LL(G,T)$
  if $likelihood - prev\_likelihood < \varepsilon$ **then return** G
  $prev\_likelihood \leftarrow likelihood$
end

The inside-outside algorithm can be sensitive to the initial assigned probabilities. The closer the initial parameters are to the best possible ones, the more likely the algorithm will not get stuck into some local maximum. Moreover, experiments have shown that certain techniques used to assign good initial parameters can on average lead to a reduction of the number of iterations by half (Lari and Young, 1990).

2.6 Bibliographical Background

PCFGs were first studied in the late 1960s and early 1970s as theoretical objects by themselves (Booth, 1969; Booth and Thompson, 1973; Salomaa, 1969; Huang and Fu, 1971) and for applications in pattern recognition (Grenander, 1967) and computational linguistics (Suppes, 1970). An early work by Horning

(1969) showed that PCFGs are theoretically learnable from positive data (under an idealised and impractical scenario, without any direct insight into how this result can be extended to practical scenarios), a result which was extended by Angluin (1988b).

The 1990s saw renewed interest in PCFGs. From the field of NLP, systems were developed using PCFGs as language models and as parsers for natural language (Stolcke, 1994; Jurafsky et al., 1995; Charniak, 1997; Johnson, 1998; Collins, 1999). In bioinformatics, PCFGs were used for RNA secondary structure prediction (Sakakibara et al., 1994b; Grate, 1995). A number of statistical properties of PCFGs were proved by (Chi, 1999; Chi and Geman, 1998), PCFGs were compared to probabilistic push-down automata in (Abney et al., 1999) and the important Inside-Outside algorithm was reviewed by (Lari and Young, 1990). Some useful algorithms for PCFG were developed in this period (Jelinek et al., 1992; Jelinek and Lafferty, 1991; Stolcke, 1995)

Recently, more algorithms that polynomially compute functions on PCFGs were given (Nederhof and Satta, 2009, 2011) and distances between PCFGs and probabilistic finite state automata were investigated (Nederhof and Satta, 2008). In NLP, more work made use of PCFGs as models for natural language (Charniak et al., 2006; Petrov et al., 2006; Cohen and Smith, 2010a). An interesting recent development is the use of PCFGs with latent annotations (Matsuzaki et al., 2005; Petrov et al., 2006), which has given positive results.

On the consistency problem, a necessary and sufficient condition for PCFG consistency was given by Booth and Thompson (1973). It has been shown that the inside-outside algorithm and the relative weighted frequency method for parameter estimation always produce consistent PCFGs (Chi, 1999; Sánchez and Benedí, 1997; Nederhof and Satta, 2006). An interesting recent development by Gecse and Kovács (2010) is an algorithm for transforming an inconsistent PCFG to a consistent one without changing the language generated by the underlying CFG.

With regards to parsing, both the CYK and Earley algorithm can be considered as chart parsers (Kay, 1996). A more general view of parsing is given in (Pereira and Warren, 1983), where parsing is compared with definite-clause deduction. Although both the CYK and Earley algorithm work in polynomial time, for practical applications where grammars are big and strings to be parsed are long, they can be very inefficient. This led to the development of more efficient parsing algorithms that work only on a subset of context-free grammars, thus sacrificing the ability to parse any CFG for more efficiency. The LR (Knuth, 1965), SLR, LALR (Deremer, 1969) and other parsers were developed for parsing (unambiguous) programming languages whilst the GLR parser (Tomita, 1985) was developed for natural language parsing.

Other methods apart from the inside-outside algorithm were developed for parameter estimation. Viterbi EM (Cohen and Smith, 2010b) is a type of expectation maximization algorithm (like inside-outside) where the most probable parse tree is used to update the probabilities. Some alternative parameter estimation methods that are not directly based on the maximum likelihood principle were also proposed. For example, Bayesian parameter estimation (Johnson et al., 2007) uses prior distributions on the possible parameters so that certain types of parameters are more favoured than others (irrespective of the sample) whilst spectral learning (Cohen et al., 2012) exploits information in the eigenvectors of matrices built from sample statistics.

3

Formal and Empirical Grammatical Inference

Grammatical inference is a sub-field of machine learning where the hypothesis space is a set of formal grammars or automata. It incorporates techniques which either:

1. Provably learn (given some formal definition of what learning means) classes of grammars or automata in well-defined scenarios (where the information given and conditions which this information must satisfy are clearly defined), or
2. Empirically learn (i.e. evaluation of learning through experiments) grammars or automata (where the classes might not be well-defined) for the purpose of modelling some natural phenomenon.

In this chapter, we explain some relevant grammatical inference algorithms which formally (Section 3.1) or empirically (Section 3.2) learn CFGs or PCFGs in polynomial time. We give five different formal results, each obtained on a different learning model and on different subclasses. For the empirical results, we focus solely on algorithms that learn grammars for modelling natural language grammars.

3.1 Formal Grammatical Inference

We classify the results obtained in formal grammatical inference of subclasses of context-free or probabilistic context-free grammars according to the learning model used. We define 5 different *efficient* learning models, where by efficient we mean that they require the learning algorithm to take polynomial time with respect to the information given. Note that three of the learning models presented here use only positive data, one uses both positive and negative data and one uses membership and equivalence queries. For each learning model, we describe one subclass of context-free grammars that is learnable under that model. These 5 results are based on the works of (Clark and Eyraud, 2007), (Higuera and Oncina, 2002), (Higuera and Oncina, 2003), (Clark, 2006) and (Clark, 2010a). We assume familiarity with basic notions of formal language theory, machine learning and statistics.

3.1.1 Polynomial Identification in the Limit from Positive Data

In this learning model, the algorithm is required to find a grammar by using only positive data and in time polynomial w.r.t. this data. Moreover, the algorithm must always return a grammar equivalent to the target one whenever the sample contains a specific set of strings, known as a characteristic sample, which should be of polynomial size w.r.t. the target grammar.

Definition 3.1.1. *(Higuera, 1997) A class of grammars $\mathscr{G}$ is **polynomially identifiable in the limit from positive data** if there exists two polynomials p(), q() and an algorithm* **A** *such that for any $G \in \mathscr{G}$:*

1. *Given a finite sample $S_+ \subseteq L(G)$ of size $m = \sum_{w \in S_+} |w|$,* **A** *returns a grammar in $\mathscr{G}$ in time $p(m)$*
2. *There exists a finite characteristic sample CS_+ of size less than $q(n)$, n being the size of G, such that if $CS_+ \subseteq S_+$ then* **A** *returns a grammar G' such that $L(G) = L(G')$*

Theorem 3.1.1. *The class of context-free grammars is not polynomially identifiable in the limit from positive data*

Proof. This follows directly from Gold's famous result in (Gold, 1967) that no super-finite class of languages is identifiable in the limit from positive data (i.e. even if we do not require the algorithm to take polynomial time, the class of context-free grammars is still not learnable under this model).

We shall prove this theorem by contradiction. Let $\mathscr{L}$ be a super-finite language class (i.e. a class of languages which contains all the finite languages and at least one infinite language. Clearly, the class of CFLs is super-finite), and let $L_{inf} \in \mathscr{L}$ be an infinite language. Assume that there exists a learning algorithm **A** that identifies $\mathscr{L}$ in the limit. We shall construct an infinite sequence of examples S_+ on which **A** does not converge to the language L_{inf}. This contradicts our assumption since, by definition, any learning algorithm that identifies in the limit a class of languages, should converge to any language in that class given any infinite sequence of examples from the language.

Let s_1 be a string such that there exists a language $L_1 \in \mathscr{L}$ which contains only the string s_1 i.e. $L_1 = \{s_1\}$. So, for **A** to identify $\mathscr{L}$ in the limit, it should be able to produce L_1 in the limit. Thus, if we keep on adding s_1 to S_+, at some point, **A** would return L_1. When this happens, we add a string s_2 to S_+, where $L_2 \in \mathscr{L}$ contains both s_1 and s_2 i.e. $L_2 = \{s_1, s_2\}$. The string s_2 is kept on added into S_+ until eventually **A** returns L_2. If this procedure continues indefinitely, S_+ will be equal to $s_1, \ldots, s_2, \ldots, s_3, \ldots$; where the number of repetitions of s_i is enough for **A** to produce $L_i = \{s_1, s_2, s_3, \ldots, s_i\}$. Note that S_+ is now an infinite sequence of strings from which **A** can learn languages L_1, L_2, L_3 etc.... Let $L_{inf} = \{s_n | n \in \mathbb{N}\}$. So, the sequence S_+ contains all the strings in L_{inf}, however, L_{inf} can never be produced by **A**. Thus, the language class $\mathscr{L}$ is not identifiable in the limit, which contradicts our assumption.

□

Definition 3.1.2. *(Clark and Eyraud, 2007) A language L is **substitutable** iff for all $u, v \in \Sigma^*$, if u and v are weakly substitutable w.r.t. L then u and v are also strongly substitutable w.r.t. L, where:*

1. *Two strings u and v are **weakly substitutable** w.r.t a language L iff there exists $l, r \in \Sigma^*$ such that $lur \in L$ and $lvr \in L$.*
2. *Two strings u and v are **strongly substitutable** w.r.t a language L iff for all $l, r \in \Sigma^*$, $lur \in L \Leftrightarrow lvr \in L$. Note that this is an equivalence relation. We shall denote the equivalence class of x by $[x]$.*

*The **class of substitutable context-free grammars** is exactly the set of all context-free grammars that generate a substitutable language.*

Theorem 3.1.2. *(Clark and Eyraud, 2007) The class of substitutable context-free grammars is polynomially identifiable in the limit from positive data*

Proof Idea: Given that the target grammar is $G =< N, \Sigma, R, S >$, the characteristic set CS_+ is $\{lwr \,|\, (A \to \alpha) \in R, (l,r) = c(A), w = w(\alpha)\}$, where $c(A)$ is the smallest pair of strings (l,r), such that $S \Rightarrow^* lAr$ and $w(\alpha)$ is the smallest string in Σ^* generated by $\alpha \in (\Sigma \cup N)^+$. Given a sample S_+, the learning algorithm simply builds a grammar $G' =< N', \Sigma, R', S' >$:
– $R' = \{\phi(x) \to \phi(y)\phi(z) \,|\, x, y, z \in Subs(S_+)\,;\, |x| > 1\,;\, x = yz\} \cup \{\phi(a) \to a \,|\, a \in \Sigma\}$

– $S' = \{\phi(w) \mid w \in S_+\}$

where ϕ is a function from $Subs(S_+)$ to N' such that if u, v are weakly substitutable w.r.t. S_+ then $\phi(u) = \phi(v)$. Each non-terminal in G' corresponds to an equivalence class of the strongly substitutable relation (i.e. the terminal yield of a non-terminal $\phi(x)$ is exactly $[x]$). Since for every non-terminal A in G, CS_+ contains at least one substring x in the terminal yield A, if $CS_+ \subseteq S_+$ then G' will have a non-terminal $\phi(x)$ corresponding to A. The production rules of G' are just all the possible logical deductive relationships (see (Clark, 2010b) for an explanation) between the equivalence classes represented by the non-terminals, which are based on the fact that if $[x] = [yz]$ then $[y][z] \subseteq [x]$ and on the trivial fact that $a \in [a]$.

□

One should note that no polynomial bound is given on the size of CS_+ w.r.t. the size of the target grammar. It is argued in (Clark and Eyraud, 2007) that this requirement is not suitable for learning context-free grammars. The following example is given to prove this point:

Example A CFG G_n with a one letter alphabet $\{a\}$ is defined as follows: it contains a set of non-terminals $N_1 \ldots N_n$. The productions consist of $N_i \rightarrow N_{i-1}N_{i-1}$ for $i = 2, \ldots n$ and $N_1 \rightarrow aa$. The sentence symbol is N_n. It can easily be seen that $L(G_n)$ consists of the single string a^{2^n}, yet the size of the representation of G_n is linear in n.

One possible solution for this problem is to require the characteristic sample to be polynomial in a parameter other than the size of the target grammar, like for example the thickness of a CFG (see (Yoshinaka, 2008) for more details).

Other works which use polynomial identification in the limit from positive data as a learning model include (Yokomori, 2003), (Laxminarayana and Nagaraja, 2003), (Yoshinaka, 2006) and (Yoshinaka, 2008).

3.1.2 Polynomial Identification in the Limit from Positive and Negative Data

This learning model is practically the same as the one described in section 3.1.1, the only difference being that the sample contains both positive and negative data.

Definition 3.1.3. *(Higuera, 1997) A class of grammars $\mathscr{G}$ is **polynomially identifiable in the limit from positive and negative data** iff there exists two polynomials p(), q() and an algorithm* **A** *such that for any $G \in \mathscr{G}$:*

1. *Given a finite sample $(S_+, S_-) \in (L(G) \times \Sigma^* \backslash L(G))$ of size $m = \sum_{w \in S_+} |w| + \sum_{w \in S_-} |w|$,* **A** *returns a grammar in $\mathscr{G}$ in time $p(m)$*
2. *There exists a finite characteristic sample (CS_+, CS_-) of size less than $q(n)$, n being the size of G, such that if $CS_+ \subseteq S_+$ and $CS_- \subseteq S_-$ then* **A** *returns a grammar G' such that $L(G) = L(G')$*

Theorem 3.1.3. *(Higuera, 1997) The class of context-free grammars is not polynomially identifiable in the limit from positive and negative data*

Proof: We prove this by contradiction by showing that if this theorem is false then the equivalence problem for CFGs will be decidable (it is widely known that the equivalence problem for CFGs is undecidable). Given two grammars G_1 and G_2, where the size n of G_1 is bigger or equal to that of G_2, we can generate all the strings in $\Sigma^{\leq q(n)}$ (intractable but still computable). Clearly, if there is a string in $\Sigma^{\leq q(n)}$ accepted by only one of the two grammars then they are not equivalent. If G_1 and G_2 accept exactly the same strings in $\Sigma^{\leq q(n)}$, then they are equivalent. This is because we are guaranteed that $S_+ = \{w \in \Sigma^{\leq q(n)} | w \in L(G_1)\}$ and $S_- = \Sigma^{\leq q(n)} \backslash S_+$ contain a characteristic sample which identifies only one language. Thus, we gave a decidable procedure for the CFG equivalence problem.

□

Definition 3.1.4. *(Higuera and Oncina, 2002) A CFG $G =< N, \Sigma, P, S >$ is **linear** if $P \subset N \times (\Sigma^* N \Sigma^* \cup \Sigma^*)$. A **deterministic linear** (DL) CFG is a linear CFG where:*

1. *all rules are of the form $T \to aT'u$ or $T \to \lambda$ and*
2. *if $T \to a\alpha$ and $T \to a\beta$ then $\alpha = \beta$*

for $a \in \Sigma$, $u \in \Sigma^*$ and $\alpha, \beta \in N\Sigma^*$.

Definition 3.1.5. *A DL CFG G is in **normal form** if:*

1. *G has no useless non-terminals*
2. *For every production rule $T \to aT'u$, u is the longest common suffix in the set of strings $\{w \in \Sigma^* | T \Rightarrow^* aw\}$*
3. *If the terminal yield of two non-terminals A and B is the same, then $A = B$.*

Theorem 3.1.4. *(Higuera and Oncina, 2002) The class of deterministic linear context-free grammars is polynomially identifiable in the limit from positive and negative data*

Proof Idea: (Higuera, 2010) The learning algorithm incrementally builds a grammar using the normal form described in definition 3.1.5. The algorithm keeps a queue of non-terminals to explore. At the beginning, the start symbol is added to the grammar and to the exploration queue. At each step, a non-terminal A is extracted from the queue and a terminal symbol a is chosen in order to continue parsing the data. From these a new rule is proposed, based on the second condition of definition 3.1.5: $A \to aA_?w$. Each time a new rule is proposed the only non-terminal that appears on its right-hand side, $A_?$, is checked for equivalence with a non-terminal in the grammar. We denote this non-terminal $A_?$ in order to indicate that it still needs to be named. If a compatible non-terminal is found, the non-terminal is named after it. If not, a new non-terminal is added to the grammar (i.e. it's promoted) and to the exploration queue. In both cases the rule is added to the grammar. By simulating the run of this algorithm over a particular grammar, a characteristic sample consisting of positive and negative examples is constructed in such a way that there is always evidence in the sample which leads to the correct decisions by the learning algorithm.

□

Other works which use polynomial identification in the limit from positive and negative data as a learning model include (Sempere and García, 1994) and (Eyraud et al., 2007)

3.1.3 Polynomial Identification in the Limit with Probability One

In this learning model it is assumed that the target grammar generates a probabilistic language using only positive data which is sampled i.i.d. in $\mathcal{O}(1)$ time (i.e. the time complexity of the sampling process is not an issue). This model requires the algorithm to return, with probability one, a grammar equivalent to the target grammar in all but a finite number of attempts. Moreover, the algorithm should work in time polynomial w.r.t. the size of the sample it has so far.

Definition 3.1.6. *(Higuera and Oncina, 2004) A class of probabilistic languages $\mathscr{L}$ is **polynomially identifiable in the limit with probability one** in terms of a class of grammars $\mathscr{G}$ if there is an algorithm **A** and a polynomial $p()$ for which for any language $L \in \mathscr{L}$, and any increasing sequence $\langle s_n \rangle$ of strings drawn according to L:*

1. *A returns grammars $G_0, G_1, \ldots G_n \ldots$ such that, with probability 1, for all but finite number of values of n, $L(G_n) = L(G)$*
2. *A works in time $p(\|\langle s_n \rangle\|)$ time.*

Definition 3.1.7. *(Higuera and Oncina, 2003) A **stochastic deterministic linear** (SDL) CFG is any probabilistic CFG whose structure is a DL CFG (see definition 3.1.4).*

Theorem 3.1.5. *(Higuera and Oncina, 2003) The class of SDL CFGs is polynomially identifiable in the limit with probability one.*

Note that the number of examples required to learn is not bounded in this learning model. This leads to a problem in the sense that the model is too weak due to the following result:

Theorem 3.1.6. *(Higuera and Oncina, 2004) Let $\mathscr{G}$ be any recursively enumerable class of grammars for which, for any sample S_+, the distance $\mathbf{L}_\infty(S_+, G)$ is computable. $\mathscr{G}$ is polynomially identifiable in the limit with probability one.*

This means that the whole class of context-free grammars is learnable under this model. An alternative model is presented in (Higuera and Oncina, 2004) that takes care of this issue. However, this model is too strong (not even subclasses of probabilistic regular grammars are learnable). An interesting line of research would be to find a new learning model along these lines which is not too weak (i.e. for which the whole class of context-free grammars is not learnable) and not too strong (i.e. for which some non-trivial subclass of context-free grammars is learnable).

3.1.4 Probably Approximately Correct Learning from Positive Data

In this learning model, the learning algorithm needs to return a grammar which, most of the time (measured by a confidence parameter δ), is correct or nearly correct (measured by an error parameter ε). This is done with polynomial bounds on the computation time and data.

Definition 3.1.8. *(Higuera, 2010) Let $\mathscr{G}$ be a class of grammars. $\mathscr{G}$ is **polynomially probably approximately correct** learnable (polynomially PAC-learnable) from positive data if there exists an algorithm $\mathbf{A}$ such that for each $n \in \mathbb{N}$, for every $G \in \mathscr{G}$ of size at most n, for every distribution $\mathscr{D}$ over $L(G)$, for every $\frac{1}{2} > \varepsilon > 0$ and $\frac{1}{2} > \delta > 0$, if $\mathbf{A}$ is given examples of size less than or equal to m sampled i.i.d. from $\mathscr{D}$ then, with probability at least $1-\delta$, $\mathbf{A}$ outputs a grammar G' such that $Pr_D(x \in L(G) \oplus L(G')) \leq \varepsilon$. Also, $\mathbf{A}$ has to work in time polynomial in $\frac{1}{\varepsilon}, \frac{1}{\delta}, |\Sigma|, m$ and n.*

Theorem 3.1.7. *The class of context-free grammars is not polynomially PAC-learnable*

Proof: If the following lemma is true, then this theorem will be true since context-free languages are not closed under intersection:

Lemma 3.1.8. *(Higuera, 2010) If a language class $\mathscr{L}$ contains two languages L_1 and L_2 such that $L_1 \cap L_2 \neq \emptyset$ and $L_1 \cap L_2 \notin \mathscr{L}$, then $\mathscr{L}$ is not PAC-learnable from positive data (i.e. not even if we allow the resource not to be polynomially bounded).*

Let $\mathscr{L} = \{L_1, L_2\}$, where $L_1 = \{w_1, w_3\}$ and $L_2 = \{w_2, w_3\}$. Consider the distribution $\mathscr{D}_1$ where $Pr_{\mathscr{D}_1}(w_1) = Pr_{\mathscr{D}_1}(w_3) = \frac{1}{2}$ and the distribution $\mathscr{D}_2$ where $Pr_{\mathscr{D}_2}(w_2) = Pr_{\mathscr{D}_2}(w_3) = \frac{1}{2}$. Learning L_1 from $\mathscr{D}_2$ and learning L_2 from $\mathscr{D}_1$ will lead to the same result. However, when testing the results, the error in one of the two cases will be at least $\frac{1}{2}$.

□

Definition 3.1.9. *(Clark, 2006) A CFG is **unambiguous** if it admits only one parse tree per string it generates. A CFG $G =< N, \Sigma, R, S >$ is **Unambiguous Non-Terminally Separated** (Unambiguous NTS) if it is unambiguous and for all $A, B \in N$, $\alpha\beta\gamma \in (\Sigma \cup N)^*$ such that $A \Rightarrow^* \alpha\beta\gamma$ and $B \Rightarrow^* \beta$ then $A \Rightarrow^* \alpha B\gamma$.*

Theorem 3.1.9. *(Clark, 2006) The class of Unambiguous NTS CFGs is polynomially PAC-learnable from positive data.*

Proof Idea: The learning algorithm here is a probabilistic version of the algorithm in (Clark and Eyraud, 2007) (explained in section 3.1.1). Instead of testing for weak substitutability, it tests for probabilistic strong substitutability. Two substrings u and v from a sample S_+ are probably strongly substitutable if the following conditions are met for some pre-defined parameters m_0, μ_2 and μ_3:

1. Both u and v appear as substrings in the sample S_+ at least m_0 times.
2. The $\mathbf{L}_\infty$ norm for the empirical context distributions of both u and v is bigger than $\frac{\mu_2}{2}$. This means that both u and v should have at least one context which appears frequently vis-a-vis the other contexts (i.e. its empirical probability should be bigger than $\frac{\mu_2}{2}$).
3. The $\mathbf{L}_\infty$ distance between the empirical context distributions of u and v should be less than $2\mu_3$. This means that u and v should share as much context in common as possible with relatively close probabilities.

It is proved in (Clark, 2006) that if all these conditions are met then, with high probability (based on the parameters m_0, μ_2 and μ_3), u and v are strongly substitutable. The rest of the algorithm proceeds with constructing the production rules exactly as is done in (Clark and Eyraud, 2007).

□

3.1.5 Learning from a Minimally Adequate Teacher

In this learning model, the learner is only given the faculty to ask two types of questions to an oracle which has access to the target grammar G:

1. Is a particular string w in $L(G)$? (Membership query)
2. Is a particular grammar G' equivalent to the target grammar? (Strong equivalence query)

The answer returned for the first question is simply a *yes* or a *no*. For the second question, either a *yes* is given as answer or a string in the symmetric distance between $L(G)$ and $L(G')$. The faculty to use these two queries is known as a Minimally Adequate Teacher (MAT).

Definition 3.1.10. *A class of grammars $\mathscr{G}$ is **Polynomially MAT-Learnable** if there exists an algorithm $\mathbf{A}$ such that, for any target grammar $G \in \mathscr{G}$, $\mathbf{A}$ returns a grammar G' equivalent to G by using a polynomial number of queries with respect to both the maximum length of the examples returned by the equivalence oracle and the number of non-terminals*

It is yet not known whether the whole class of context-free grammars is polynomially learnable by a MAT.

Definition 3.1.11. *(Clark, 2010a) A CFG $G =< N, \Sigma, P, S >$ is **congruential** if for every $A \in N$, $u, v \in \Sigma^*$, if $A \Rightarrow^* u$ and $A \Rightarrow^* v$ then u and v are strongly substitutable w.r.t. $L(G)$ (see definition 3.1.2).*

Theorem 3.1.10. *(Clark, 2010a) The class of congruential context-free grammars is polynomially MAT-Learnable*

Proof idea: Consider a non-empty finite set of strings K (where the empty string λ is always contained in K) and a non-empty finite set of contexts F. Now consider an observation table with rows indexed by strings from KK, columns indexed by contexts from F and cells with either 0,1 or ?. A cell from a row indexed by w and a column indexed by (l,r) has a value of 0 if lwr is known to not be in the target language, has a value of 1 if lwr is known to be in the target language and has a value of ? if it is not known whether lwr is in the target language or not. An observation table is complete whenever all cells are either 0 or 1. An observation table is consistent if for all u_1, u_2, v_1, v_2 in K, if $l_1u_1r_1$, $l_1u_2r_1$, $l_2v_1r_2$, $l_2v_2r_2$ are known to be in the target language (i.e. their cell has a value of 1), then there should be a cells for $l_3u_1u_2r_3$ and $l_3v_1v_2r_3$ for some context (l_3,r_3). A CFG can be constructed from a complete and consistent table using a similar idea to the one used in (Clark and Eyraud, 2007) (see (Clark, 2010a) for more details).

The following is a general idea of the learning algorithm:

1. Start with $K = \lambda$ and $F = \{(\lambda,\lambda)\}$.
2. Construct the CFG from the observation table
3. Submit it to the equivalence query
4. If rejected, use the counter-example given by the oracle to update the observation table
5. Use membership queries to fill the 'gaps' in the table so that it is complete. Use also these queries to make sure that the table is consistent.
6. Return to step 2.
7. If the equivalence query in step 3 is accepted, return this grammar as the answer.

It is proved in (Clark, 2010a) that the algorithm terminates in polynomial time with respect to the size of the strings returned by the equivalence query and with respect to the size of K.

Note that this learning algorithm is an extension of Angluin's well-known L* algorithm (Angluin, 1987).

□

Other works which polynomially learn subclasses of context-free grammars using a MAT include (Ishizaka, 1990) and (Shirakawa and Yokomori, 1993).

3.2 Empirical Grammatical Inference

In contrast with formal grammatical inference, in empirical grammatical inference no proof is given that the learning algorithm has somehow learned something (Higuera, 2010). Moreover, there is no need of a learning model or a well-defined hypothesis space. In empirical GI, the only concern is to learn good grammars from the information given, where the goodness of a grammar is evaluated through some sort of experiment. Experiments vary according to the application domain.

In this section, we review some results from the literature that can be categorized under empirical GI. We focus exclusively on systems which were aimed at inferring CFGs or PCFGs for the NLP domain. Some have also been applied in other domains (Sakakibara et al., 1994a; Moore and Essa, 2002; Chen et al., 2008), but all of the systems we explain were mainly evaluated for NLP purposes. We also restrict our attention to systems which use positive data only. These are systems which are capable to learn using only the strings (or sentences in NLP) from the target language, without any structural information or any other type of information.

We start this section by first explaining issues from the NLP setting (Section 3.2.1). We explain the NLP problems where Empirical GI can be applied to, the types of information these problems take as input and the ways in which they are evaluated. We then classify these systems as follows:

– MDL based: Systems which try to infer small grammars that compactly encode the given data. In such systems, the generalization from the given data is driven by the trade-off between the grammar complexity and the complexity of the data given the grammar (Stolcke, 1994; Petasis et al., 2004) (Section 3.2.2).

– Distributional based: Systems that exploit the relationship between contexts and substrings. In such systems, the generalization from the given data is driven by similarity metrics between substrings, depending on the similarity of their contexts (Adriaans et al., 2000; van Zaanen, 2001) (Section 3.2.3).

– State-of-the-art: Systems which currently give the best results for unsupervised parsing (Klein, 2004; Bod, 2006a; Seginer, 2007) or language modelling (Solan et al., 2005).

3.2.1 NLP Setting

We consider two NLP problems which can be solved by PCFG inference:

1. Syntactic analysis: The problem of assigning structural descriptions to sentences, where these descriptions syntactically explain the sentence (Clark et al., 2010). There are several types of structural descriptions, but we will focus exclusively on phrase structure trees (which take the form of CFG parse trees).
2. Language modelling: The problem of finding a model that is able to compute the probability of a given sentence and/or the probability that a particular word will occur next in a sentence (Clark et al., 2010). With PCFGs as models, we can compute both.

Input

The following are different kinds of inputs for the above problems:

1. Raw sentences which are readily segmented into words.
2. Same as (1) + a part-of-speech (POS) tag for every word.
3. A treebank, which consists of sentences annotated with their syntactic structure in the form of parse trees. A treebank naturally includes information (1) and (2)
4. raw/tagged sentences + a small treebank.

For many natural languages, there is a large availability of raw sentences. Although manually tagged sentences are much less available due to the infeasibility of manual tagging, very good automatic tagging systems exists for major languages. On the other hand, treebanks are hard to find and build, and automatic generation of treebanks is essentially the syntactic analysis problem we need to solve in the first place.

The most useful setting (and the hardest to work with) is when only raw or tagged sentences are used as input. This is known as the unsupervised setting (and the syntactic analysis problem in the unsupervised setting is known as unsupervised parsing). In this work, we focus exclusively on this setting. Whilst a lot of work has been done in the other settings (i.e. the supervised setting with (3) as input and semi-supervised setting with (4) as input) where good results have been obtained, work in unsupervised parsing still lags behind and the current state-of-the-art results are still incomparable with the other settings (Clark et al., 2010). Thus, more effort is needed in the unsupervised setting, where any good results obtained in this setting will supersede results from other settings.

Evaluation

Regarding the evaluation of the language modelling problem, there are two ways of doing this in the unsupervised setting. If we have access to the target grammar, then evaluation boils down to computing distance or similarity metrics between the probabilistic language of the target grammar and the probabilistic language of the learned grammar. The smaller the distance between the two, the better the learned grammar is. However, we face the problem that many distance or similarity metrics for PCFGs are undecidable (see Chapter 4 for our contribution to this problem). So, an alternative is to compute estimated distances by sampling a lot of strings from the two grammars and calculate the distance between the probabilities of these strings.

If on the other hand no target grammar is available, then perplexity or other similar measures are used for evaluation. The perplexity of a grammar on a sample measures how well the grammar's distribution predicts the sample; the lower the perplexity, the better the grammar's distribution is. A test sample S (disjoint from the training sample) is used to calculate the perplexity PP of a learned grammar G as follows:

$$PP(G,S) = \sqrt[n]{\prod_{w \in S} Pr_G(w)^{cnt(w,S)}}$$

where n is the number of strings in S and $cnt(w,S)$ is the number of strings w in S.

Evaluation for unsupervised parsing is more problematic (van Zaanen and Geertzen, 2008). First of all, at least a small treebank is needed as a test sample to evaluate the quality of the learned structures. Then there are different ways to compare structures. There are essentially two important features that can be measured here:

1. The accuracy of the constituents. Note that a constituent is any occurrence of a substring (or a sequence of words in NLP) that is dominated by a non-terminal. Since parse trees can be represented in bracketed format, then the constituents are simply the substrings (or phrases) which are enclosed within brackets.
2. The accuracy of the non-terminal labels.

Non-terminal labels are tricky to assess since one has to find a mapping between the learned and the target labels. We avoid this problem, as many others do in the literature (van Zaanen, 2001; Klein, 2004; Bod, 2006b; Seginer, 2007; Bordag et al., 2008), by simply ignoring these labels. The standard way of evaluating constituent accuracy is to use the precision and recall metrics on the bracketings. The precision and recall on one particular sentence can be easily calculated as follows:

$$Precision = \frac{TP}{TP+FP}$$

$$Recall = \frac{TP}{TP+FN}$$

where TP are the true positives (i.e. correctly induced brackets), FP are the false positives (i.e. induced brackets which are not in the target bracketing) and FN are the false negatives (i.e. target brackets which were not induced). There are then two ways to calculate the average precision and recall over a test-sample: micro and macro precision/recall.

$$Micro\text{-}Precision = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + FP_i}$$

$$Micro\text{-}Recall = \frac{\sum_{i=1}^{n} TP_i}{\sum_{i=1}^{n} TP_i + FN_i}$$

$$Macro\text{-}Precision = \frac{1}{n}\left(\sum_{i=1}^{n} \frac{TP_i}{TP_i + FP_i}\right)$$

$$Macro\text{-}Recall = \frac{1}{n}\left(\sum_{i=1}^{n} \frac{TP_i}{TP_i + FN_i}\right)$$

where TP_i, FP_i and FN_i are the true positives, false positives and false negatives for the i^{th} sentences in a test sample made up of n sentences. We can take the harmonic mean of the precision and recall values to obtain a single value (known as the F-Score) as follows:

$$F - Score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

In practice, certain type of brackets may not be taken into consideration due to their triviality. These typically are the empty brackets, those which span the whole sentence and those which span only one word).

3.2.2 MDL-based systems

A learning algorithm which takes only positive data as input has to cater for both the undergeneralisation and overgeneralisation problems. This is because there always exists two 'extreme' undesirable grammars consistent with any given positive data. One is an undergeneralising grammar G_u that only generates the training sample and nothing else. The other is an overgeneralising grammar G_o that generates Σ^*. Both G_u and G_o are consistent with any given positive training sample.

To overcome this problem, MDL-based systems direct their search towards grammars that minimize the sum of the following two measures:

1. The size of the grammar
2. The size of an encoded training data given the grammar

By minimizing the size of the grammar, grammar G_u is avoided since this grammar will be large (in fact, the size of G_u will be roughly as big as the training sample). On the other hand, G_o is avoided because the size of the encoded training data given G_o will not be any smaller than simply the size of the training data itself.

Apart from avoiding the two 'extremes', a grammar which is relatively small *and* is a close fit to the data follows the following two principles simultaneously:

1. Occam's razor principle: Among competing grammars consistent with the data, the more compact grammars should be chosen.
2. Maximum Likelihood principle: Among competing grammars consistent with the data, the grammars which are more likely to generate the data should be chosen.

Therefore, MDL-based systems try to find grammars which have a good trade-off between these two intuitively desirable properties. Moreover, there are also well-founded theoretical reasons for preferring grammars that follow these two principles (Kearns and Vazirani, 1994; Grünwald, 2007). In this section, we

briefly describe two MDL-based systems, Bayesian Model Merging (Stolcke, 1994) and e-GRIDS (Petasis et al., 2004).

Bayesian Model Merging

Stolcke (1994) describes a method for inferring PCFGs involving the following three steps:

1. *Data incorporation*: An initial Weighted CFG (i.e. a CFG with counts assigned to rules instead of probabilities) is built that generates only the training sample. This is done by simply adding rules of the form

$$S \rightarrow N_{a_{i,1}} N_{a_{i,2}} \ldots N_{a_{i,k}} (c)$$

and

$$N_{a_{i,1}} \rightarrow a_{i,1}(c) \qquad N_{a_{i,2}} \rightarrow a_{i,2}(c) \qquad \ldots \qquad N_{a_{i,k}} \rightarrow a_{i,k}(c)$$

for every string $w_i = a_1 a_2 \ldots a_k$ in the sample $\{w_1 \ldots w_n\}$, where c is the number of strings w_i in the sample. Therefore, $N_{a_{i,j}}$ is a unique non-terminal for the j^{th} symbol of the i^{th} string in the sample.

2. *Structure Merging*: A hill-climbing search for a better structure is performed by incrementally generalizing from the initial grammar using two operators: non-terminal merging and non-terminal chunking. Two non-terminals A and B are merged by a simple substitution of all the occurrences of A with B (or B with A). Chunking involves the substitution of a sequence of non-terminals α in the RHS of rule with a newly added non-terminal A, and the addition of the rule $A \rightarrow \alpha$. Note that a chunking operation does not generalize the grammar, however it changes the structure of the grammar's parse trees and combined with merging operations, it can produce generalized grammars which cannot be found with only merging operations. After each merge or chunk the counts of the rules are updated accordingly.
3. *Parameter Estimation*: The inside-outside algorithm is used to estimate the probabilities of the obtained CFG after structure merging.

Merging and chunking decisions are taken following maximum a-posteriori estimation, where the chosen decision is the one which returns a grammar G_{MAP} that maximizes the probability of the grammar given the data. Using Bayes' rule:

$$\begin{aligned} G_{MAP} &= argmax_G Pr(G|D) \\ &= argmax_G \frac{Pr(D|G) \cdot Pr(G)}{Pr(D)} \\ &= argmax_G Pr(D|G) \cdot Pr(G) \end{aligned}$$

$Pr(D|G)$ is simply the likelihood of the sample given the grammar, which can be calculated from the sample counts. $Pr(G)$ is a prior distribution on the grammars that favours small grammars. It is decomposed into two probabilities, one which takes into consideration only the structure of G and the other which is calculated on the parameters of G as follows:

$$Pr(G) = Pr(G_S) \cdot Pr(\theta_G|G_S)$$

$Pr(G_S)$ is a description length-induced distribution and $Pr(\theta_G|G_S)$ is a product of Dirichlet distributions, one for each non-terminal.

Instead of using a totally greedy search algorithm where only the locally best merge/chunk decision is taken, Stolcke (1994) uses a beam search that takes multiple good decisions in parallel and keeps multiple hypothesis grammars until no improved grammars are produced.

Experiments conducted by (Stolcke, 1994) on test grammars from (Cook et al., 1976) resulted in positive results; for all cases, the system managed to find the target grammars (i.e. not simply any grammar equivalent to the target). An experiment on the palindrome language returned a grammar equivalent to the target with practically the same structure. Experiments on artificial natural language grammars also returned equivalent grammars to their target, although in some cases the learned rules were quite different from the target rules. A brief experiment was done on a 1200 sentence corpus. Although some good lexical and phrasal categories were learned, the resulting grammar undergeneralized a lot.

e-GRIDS

e-GRIDS (Petasis et al., 2004) is an MDL-based heuristic system which develops on the GRIDS (GRammar Induction Driven by Simplicity) system (Langley and Stromsten, 2000) which in turn was based on the SNPR system (Wolff, 1978). It efficiently learns CFGs from positive data.

The learning strategy of e-GRIDS is basically the same as the one used in Bayesian Model Merging (Stolcke, 1994). The same merge operator and a similar chunking operator is used, and these are applied through a beam search. One additional operator is used, but this can be simulated using one chunk followed by one merge operation.

The real difference is in the way the MDL principle is applied. In (Stolcke, 1994), the MDL principle is applied using a Bayesian inference approach whilst in (Petasis et al., 2004), the same principle is applied using an information theory approach. In (Petasis et al., 2004), decisions are taken so as to minimize as much as possible the sum between the grammar description length (GDL) and the derivations description length (DDL). The GDL is the bits needed to encode a grammar G and transmit it to a receiver. The DDL is the bits needed to encode and transmit the training sample provided that the receiver knows G.

The experiments done on e-GRIDS are similar to those done by Stolcke (1994). The results obtained are also close to each other.

3.2.3 Distributional Learning

Distributional learning techniques for inducing CFGs are based on the idea that a learning algorithm can generalize from a given sample by grouping substrings (or phrases in NLP) according to the context in which they appear (Clark et al., 2010). For example, consider the following sentences in a sample:

The old dog slept
The cat that saw the big dog ran away
The cat that heard the dog ran away
The small mouse slept
the dog that heard the cat slept

The following phrases in the sample can be grouped together on the basis of having similar contexts as follows:

[*big*, *small*, *old*] [*cat*, *dog*, *mouse*] [*saw*, *heard*] [*ran away*, *slept*]
[that saw the big dog, that heard the dog, that heard the cat]
[the old dog, the cat, the big dog, the dog, the small mouse]

New sentences can be created through substitution of phrases from the same group. The following are examples of new sentences not in the sample created from one or more substitutions of phrases from sample sentences. Note that this process of substitution can potentially yield an infinite number of sentences:

the cat ran away
the big old dog saw the small old cat
the small mouse that heard the big cat that saw the old dog ran away

CFGs generating such sentences can be built from the groups. The non-terminals of such grammars generate phrases from the same group and the production rules simply represent the concatenation relationships between groups. For example, if non-terminal A generates strings from the group that contains the phrase *the old dog*, and non-terminal B generates strings from the group that contains the word *slept*, and the phrase *the old dog slept* is contained in the group of phrases generated from S, then it follows that a rule $S \rightarrow AB$ makes sense. From this rule we can then generate sentences which are not in the sample.

So, the three key problems that a distributional learning algorithm needs to tackle are:

1. Which substrings are to be grouped together?
2. Which groups of substrings should correspond to non-terminals?
3. Which concatenation relationships between groups should be used as production rules?

In this section, we very briefly describe two well-known distribution based empirical learning algorithms: EMILE (Adriaans et al., 2000) and ABL (van Zaanen, 2001).

EMILE

In EMILE (Adriaans et al., 2000), grouping of substrings is done using a context/expression expression. This is simply a boolean matrix where rows and columns correspond to all the possible substrings and contexts respectively in the sample. Element in row w and column (l,r) is true if the string lwr is in the sample, otherwise it is false. Table 3.1 is an example context/expression matrix for the sample { *'Mary likes tennis'*, *'Mary plays tennis'* }. Substrings which form a maximum-sized complete block (as shown in table 3.2) of true values are grouped together.

In reality, A very large sample is need in general in order to have complete blocks. Thus, EMILE weakens this condition by also grouping substrings which have *nearly* complete blocks (depending on a threshold parameter given as input to the algorithm). Also, EMILE starts by clustering short expressions and contexts (considered as characteristic expressions/contexts). Longer expressions are then clustered together based on the assumption that if they appear with a characteristic expression/context, then they should form part of the same group. Rules are then built by searching characteristic expressions that occur in longer expressions. For example, from the following two groups of expressions:

$[A] = \{$ *James*, *Christine*, *Mary* $\}$
$[B] = \{$ *tennis*, *football* $\}$
$[S] = \{$ *James likes tennis*, *Christine thinks that Mary likes tennis* $\}$

EMILE extracts the following rules:

$[S] \rightarrow [A]$ *likes* $[B] \mid [A]$ *thinks that* $[S]$
$[A] \rightarrow$ *James* | *Christine* | *Mary*
$[B] \rightarrow$ *tennis* | *football*

	ε likes tennis	Mary tennis	Mary likes ε	ε tennis	Mary ε	ε ε	ε plays tennis	Mary plays ε
Mary	×						×	
likes		×						
tennis			×					×
Mary likes				×				
likes tennis					×			
Mary likes tennis						×		
plays		×						
Mary plays				×				
plays tennis					×			
Mary plays tennis						×		

Table 3.1: Context/expression matrix

	Mary tennis	Mary football	Mary skiing	Mary likes ε	Mary plays ε	Mary is ε
plays	(1)	(1)				
likes	(1)	(1)	(2)			
is			(2)			
skiing					(3)	(3)
tennis				(4)	(4)	
football				(4)	(4)	

Table 3.2: Context/expression matrix with blocks

Alignment-based learning

ABL (van Zaanen, 2001) is a distributional learning algorithm designed for unsupervised parsing. The aim of this algorithm is to simply assign one parse tree for each sentence given in the training sample. Consequently, a grammar can be easily read of from these trees.

ABL is based on the principle that substitutable substrings are good candidates as constituents and should have the same type (i.e. should be derivable from the same non-terminal). The algorithm consists of two steps:

1. An alignment phase, where candidate constituents are chosen based on the similarities between aligned sentences. No potential constituent is rejected because it overlaps with a previously selected constituent.
2. A selection phase, where

For the alignment phase, ABL goes through the following procedure for every pair of strings (w_1,w_2) in the sample

1. Align w_1 to w_2
2. Find the identical parts of sentences w_1 and w_2 and linking them
3. Group adjacent linked words
4. Group the remaining adjacent non-linked words
5. Assign non-terminals to the candidate constituents (where the constituents are the groups of distinct parts)

Identical parts of the sentences are found using a string edit distance algorithm, which finds the minimum number of edit operations (insertion, deletion and substitution) to change one sentence into the other. The identical words in a sentence are those found in places where no edit operation is applied. Clearly, this process is ambiguous (i.e. can give different results). For example, given the 2 sentences: *"from Malta to Italy"* and *"from Italy to Malta"*, the following are all possible results (underlined words are the linked identical words and the words in brackets are the candidate constituents):

from $()_1$ Malta (to Italy)$_2$
from (Italy to)$_1$ Malta $()_2$

from (Malta to)$_1$ Italy $()_2$
from $()_1$ Italy (to Malta)$_2$

from (Malta)$_1$ to (Italy)$_2$
from (Italy)$_1$ to (Malta)$_2$

To solve this ambiguity problem, the algorithm uses one of these three methods:

1. Select the alignment with the maximum number of linked words. When multiple alignments admit the same maximum number of linked words, the choice between them is postponed to the selection stage of the algorithm.
2. Select one alignment at random.
3. Keep all possible alignments, and decide in the selection stage of the algorithm (thus increasing the burden on the selection phase).

Another potential problem (apart from ambiguity) is that overlapping constituents can be found. The following example shows a sentence (B) aligned to two different sentences (A and C), producing overlapping constituents (*"Give me all flights"* overlaps with *"all flights from Malta to Italy"*)

Sentence A: (Book this flight)$_1$ from Malta to Italy
Sentence B: (Give me all flights)$_1$ from Malta to Italy

Sentence B: Give me (all flights from Malta to Italy)$_2$
Sentence C: Give me (your passport number)$_2$

This is where the selection phase is need. This step solves this overlapping problem by choosing constituents based on either one of the following methods:

1. The easiest method is **ABL:incr**, which assumes that the constituent identified first is the correct one. Thus, if at some point a constituent is found that overlaps a previously found constituent, then the new one is ignored. However, once an incorrect constituent is learned, it will never be corrected van Zaanen (2001).
2. Another method is **ABL:leaf**, which selects the most probable constituent. The probability of a constituent is computed using the following equation:

$$P_{leaf}(c) = \frac{|c' \in C : yield(c') = yield(c)|}{|C|}$$

where C is the set of all candidate constituents and the yield of a constituent is the list of words derived from the constituent's non-terminal.

3. The **ABL:branch** method builds on ABL:leaf by taking into account the constituent's non-terminal, NT(c).

$$P_{leaf}(c \mid NT(c) = N) = \frac{|c' \in C : yield(c') = yield(c) \wedge NT(c') = N|}{|c'' \in C : NT(c'') = N|}$$

ABL was evaluated on the AVIS and OVIS corpus (van Zaanen and Adriaans, 2001). Tables 3.3 and 3.4 show the results obtained.

		UR	UP	F
ATIS	EMILE	16.814 (0.687)	**51.588** (2.705)	25.351 (1.002)
	ABL	**35.564** (0.020)	43.640 (0.023)	**39.189** (0.021)
OVIS	EMILE	36.893 (0.769)	49.932 (1.961)	41.433 (3.213)
	ABL	**61.536** (0.007)	**61.956** (0.008)	**61.745** (0.007)

Table 3.3: Results: UR = unlabelled recall, UP = unlabelled precision, F = F-score (from van Zaanen and Adriaans (2001))

		CB	0 CB	$\leq$2 CB
ATIS	EMILE	16.814 (0.687)	**51.588** (2.705)	25.351 (1.002)
	ABL	**35.564** (0.020)	43.640 (0.023)	**39.189** (0.021)
OVIS	EMILE	36.893 (0.769)	49.932 (1.961)	41.433 (3.213)
	ABL	**61.536** (0.007)	**61.956** (0.008)	**61.745** (0.007)

Table 3.4: Results: CB = average crossing brackets, 0 CB = no crossing brackets, $\leq$2 CB = two of fewer crossing brackets (from van Zaanen and Adriaans (2001))

3.2.4 State of the art systems

Results in unsupervised parsing were very discouraging up until the beginning of the last decade. Taking the results obtained on the WSJ as an example, no system had reported better performance on the WSJ10 corpus than the simple baseline of assigning right branching parse trees (Klein, 2004). Parameter search approaches seemed unpromising back then (Lari and Young, 1990; Carroll and Charniak, 1992), so many opted for structure search. All the distributional learning and MDL-based systems we described earlier adopt a structure search strategy.

Renewed interest in parameter search algorithms started after (Klein and Manning, 2002) reported very good results in unsupervised parsing. According to its authors, CCM was the first system to surpass this right-branching baseline on WSJ10 (using POS tagged sentences) (Klein, 2004). What CCM does differently from ABL and EMILE is that it does not involve direct structure learning. CCM does not go through the given data and compare substrings and, depending on their contexts, decides which substrings should be constituents and of what type. CCM simply uses an extension of a standard parameter estimation algorithm, Expectation Minimization (EM), in order to estimate probabilities $Pr(B|S,\phi)$ (i.e. the probability that a bracketing B is used given sentences S and parameters ϕ) for a simple generative model. The most probable constituents are then chosen. The innovation with CCM was the definition of an effective generative model, the parameters used and how these were estimated.

Inspired by CCM, other parameter search systems were developed and reported marginal improvements (Bod, 2006a,b; Seginer, 2007; Bordag et al., 2008). A common theme with these systems is that they

initially consider all possible structures (in most cases, all the possible binary parse trees). On the other hand, the trend in structure learning systems is to start from nothing and incrementally find structure in sentences. This duality between 'starting with no structure' vs. 'starting with all structures' has been investigated by (van Zaanen and van Noord, 2012). Overall, it seems that better results in unsupervised parsing can be obtained by first considering all the possible structures, and then slowly perform pruning (in a structure search scenario) or estimate model parameters that maximize the likelihood of the data (in a parameter search scenario).

ADIOS (Solan et al., 2005) is one of the state-of-the-art GI systems in language modelling (Waterfall et al., 2010). It represents the given data as a directed psuedograph (loops and multiple edges are allowed, see figure 3.1 for an example), where each sentence is stored as a path in this graph. The algorithm iteratively modifies this graph by performing two tasks (which correspond to finding which phrases should be constituents and which constituents/word should be considered of the same type):

1. Subpaths of this graph (representing phrases) that are shared by a significant number of aligned paths are considered as important patterns. These patterns are added as new lexicon items. This is essentially a substitution phase in order to treat constituent phrases as units on their own (similarly to what is done in (Clark, 2001)). This step is responsible for finding structure in sentences.
2. Nodes with similar contexts (i.e. with similar incoming and outgoing edges) are merged together. This corresponds to POS tagging between words and constituent labelling between phrases. This step is responsible for generalizing from the given sentences.

This process is repeated on until no other significant patterns are found. The final graph can then be used as a generator of new sentences or transformed into a context-free grammar.

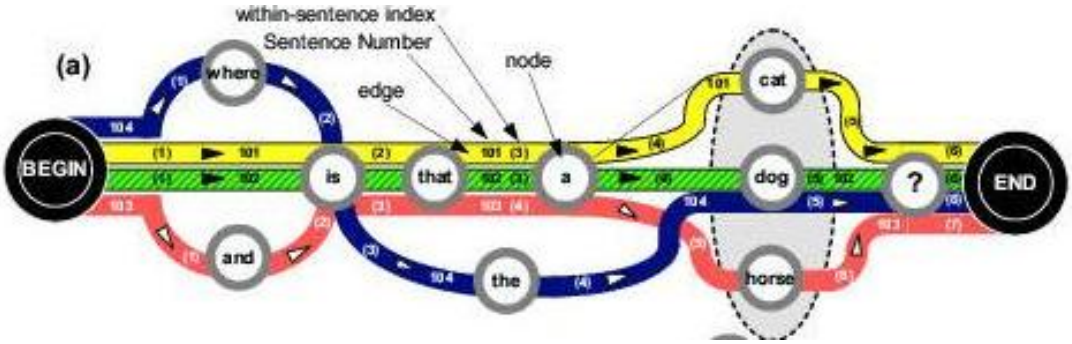


Figure 3.1: Graph built by ADIOS for the sentences *where is the dog?*, *is that a cat?*, *is that a dog?*, *and is that a horse?* (this figure is taken from (Solan et al., 2005))

Contributions

4

On the Computation of Distances for PCFGs

For any statistical model M describing probability distributions (such as PCFGs), it is interesting to ask whether it is possible to efficiently compute the distance (or an approximation of the real distance) between two probability distributions from their descriptions using M. This is an especially relevant question for the task of inducing statistical models. Having a distance function between models can be used to assess how different hypothesized models are from each other, which can be useful during the learning process. Moreover, some form of distance function is needed to evaluate how good an induced grammar is compared to a known target grammar.

The issue is that the problem of computing distances between PCFGs has barely been investigated. Therefore, as a minor contribution of this thesis, we investigate this problem and show that several distances between PCFGs are uncomputable. We also give a new insight on the open problem of PCFG equivalence, where we show that this problem is equivalent to the Chebyshev distance problem. As a positive result, we show that the problem of finding the consensus string (i.e. the most probable string) for a PCFG is computable.

This chapter is adapted from the paper *On the Computation of Distances for Probabilistic Context-Free Grammars* authored by Colin de la Higuera, James Scicluna and Mark-Jan Nederhof, which is archived on ArXiv.

4.1 Background

In the many areas where PCFGs are used as modelling tools, the following is an important question asked: *given two PCFGs, are they equivalent?*. Two PCFGs are equivalent when every string has identical probability in each distribution. More generally, a distance between distributions expresses how close they are, with a zero distance coinciding with equivalence.

When learning PCFGs, comparison between states or non-terminals often determines if a merge or a generalization is to take place. Key grammatical inference operations (Higuera, 2010) will depend on the precision with which an equivalence is tested or a distance is measured.

In the case of probabilistic finite automata, these questions have been analysed with care. The initial study by Balasubramanian (1993) showed that the equivalence problem for hidden Markov models admitted a polynomial algorithm. Later, a number of authors showed that the Euclidian distance could be computed in

polynomial time (Lyngsø et al., 1999). This important result made use of the key concept of co-emission. A similar result was obtained for PFAs (Murgue and Higuera, 2004).

Negative results were also obtained: The L1 distance, and the variation distance were proved to be intractable (Lyngsø and Pedersen, 2001). Relative entropy (or Kullback-Leibler divergence) cannot be computed between general PFAs, but it can be computed for deterministic (Carrasco, 1997) and unambiguous (Cortes et al., 2008) PFAs.

A related problem, that of computing the most probable string (also called the *consensus string*) was proved to be an NP-hard problem (Casacuberta and Higuera, 2000; Lyngsø and Pedersen, 2002); heuristics and parameterized algorithms have been proposed (Higuera and Oncina, 2013b). Some of these results and techniques were also extended to PCFGs (Higuera and Oncina, 2013a).

The co-emission of a PCFG and a hidden Markov model was discussed by (Jagota et al., 2001), who formulated the problem in terms of finding the solution to a system of quadratic equations; for the difficulties in solving such systems, see also (Etessami and Yannakakis, 2009a). By a related construction, a probabilistic finite automaton can be obtained from a given unambiguous (non-probabilistic) finite automaton and a given PCFG, in such a way that the Kullback-Leibler divergence between the two probability distributions is minimal (Nederhof and Satta, 2004).

The same problems have been far less studied for PCFGs: the equivalence problem has recently been proved (Forejt et al., 2014) to be as hard as the multiple ambiguity problem for context-free grammars: do two context-free grammars have the same number of derivation trees for each string? Since on the other hand it is known (Abney et al., 1999) that probabilistic pushdown automata are equivalent to PCFGs, it follows that the equivalence problem is also interreducible with the multiple ambiguity problem. Before that, the difficulty of co-emission and of related results was shown in (Jagota et al., 2001).

Since computing distances is at least as hard as checking equivalence (the case where the distance is 0 giving us the answer to the equivalence question) it remains to be seen, for PCFGs, just how hard it is to compute a distance, and also to see if all distances are as hard. This is the subject of this work. We have studied a number of distances over distributions, including the L1 and L2 norms, the Hellinger and the variation distances and the Kullback-Leibler divergence. We report that none of these can be computed.

On the other hand, the fact that the consensus string can be computed allows to show that the Chebyshev distance (or L_∞ norm) belongs to the same class as the multiple ambiguity problem for context-free grammars and the equivalence problem for PCFGs.

In Section 4.2 we remind the reader of the different notations, definitions and key results we will be needing. In Section 4.3 we go through the new results we have proved. We conclude in Section 4.4.

4.2 Definitions and notations

4.2.1 About co-emissions

Co-emission has been identified as a key concept allowing, in the case of hidden Markov models or probabilistic finite-state automata, computation in polynomial time of the distance for the L2 norm (and more generally any Lp norm, for even values of p): the distance can be computed as a finite sum of co-emissions.

Definition 4.2.1. *The coemission of two probabilistic grammars G_1 and G_2 is the probability that both grammars G_1 and G_2 simultaneously emit the same string:*
$\text{CoEm}(G_1, G_2) = \sum_{x \in \Sigma^*} Pr_{G_1}(x) \cdot Pr_{G_2}(x)$

A particular case of interest is the probability of twice generating the same string when using the same grammar: Given a PCFG G, the *auto-coemission* of G, denoted as $\text{AC}(G)$, is $\text{CoEm}(G,G)$. If the grammars are ambiguous, internal factorization is required in the computation of co-emission. In order to detect this we introduce the *tree-auto-coemission* as the probability that the grammar produces exactly the same left-most derivation (which corresponds to a specific tree):

Definition 4.2.2. *Given a PCFG G, the* tree-auto-coemission *of G is the probability that G generates the exact same left-most derivation twice. We denote it by* $\text{TAC}(G)$.

Note that the tree-auto-coemission and the auto-coemission coincide if and only if G is unambiguous:

Proposition 4.2.1. *Let G be a PCFG.*
$\text{AC}(G) \geq \text{TAC}(G)$.
$\text{AC}(G) = \text{TAC}(G) \iff$ *G is unambiguous.*

4.2.2 Distances between distributions

A PCFG defines a distribution over $\Sigma^\star$. If two grammars can be compared syntactically, they can also be compared semantically: do they define identical distributions, and, if not, how different are these distributions?

Definition 4.2.3. *The L1 distance (or Manhattan distance) between two probabilistic grammars G_1 and G_2 is:*

$$d_{\text{L1}}(G_1,G_2) = \sum_{x\in\Sigma^*} |Pr_{G_1}(x) - Pr_{G_2}(x)|$$

Definition 4.2.4. *The L2 distance (or Euclidian distance) between two probabilistic grammars G_1 and G_2 is:*

$$d_{\text{L2}}(G_1,G_2) = \sqrt{\sum_{x\in\Sigma^*} (Pr_{G_1}(x) - Pr_{G_2}(x))^2}$$

L2 distance can be rewritten in terms of coemission, as:

$$d_{\text{L2}}(G_1,G_2) = \sqrt{\text{CoEm}(G_1,G_1) - 2\text{CoEm}(G_1,G_2) + \text{CoEm}(G_2,G_2)}$$

Definition 4.2.5. *The L_∞ distance (or Chebyshev distance) between two probabilistic grammars G_1 and G_2 is:*

$$d_{\text{L}_\infty}(G_1,G_2) = \max_{x\in\Sigma^*} |Pr_{G_1}(x) - Pr_{G_2}(x)|$$

Note that the L_∞ distance seems closely linked with the consensus string, which is the most probable string in a language:

Definition 4.2.6. *The* variation distance *between two probabilistic grammars G_1 and G_2 is:*

$$d_V(G_1,G_2) = \max_{X\subset\Sigma^*} \sum_{x\in X} (Pr_{G_1}(x) - Pr_{G_2}(x))$$

The variation distance looks like d_{L_∞}, but is actually connected with d_{L1}:

$$d_V(G_1,G_2) = \frac{1}{2} d_{L1}(G_1,G_2) \tag{4.1}$$

A number of distances have been studied elsewhere (for example (Jagota et al., 2001; Cortes et al., 2007)):

Definition 4.2.7. *The* Hellinger distance *between two probabilistic grammars G_1 and G_2 is:*

$$d_H(G_1,G_2) = \frac{1}{2} \cdot \sum_{w \in \Sigma^*} \left(\sqrt{Pr}_{G_1}(w) - \sqrt{Pr}_{G_2}(w) \right)^2$$

The Jensen-Shannon (JS) *distance between two probabilistic grammars G_1 and G_2 is:*

$$d_{JS}(G_1,G_2) = \sum_{x \in \Sigma^*} \left(Pr_{G_1}(x) \log \frac{2Pr_{G_1}(x)}{Pr_{G_1}(x) + Pr_{G_2}(x)} + Pr_{G_2}(x) \log \frac{2Pr_{G_2}(x)}{Pr_{G_1}(x) + Pr_{G_2}(x)} \right)$$

The chi-squared (χ^2) *distance between two probabilistic grammars G_1 and G_2 is:*

$$d_{\chi^2}(G_1,G_2) = \sum_{x \in \Sigma^*} \frac{(Pr_{G_1}(x) - Pr_{G_2}(x))^2}{Pr_{G_2}(x)}$$

The Kullback-Leibler divergence, or relative entropy is not a metric:

Definition 4.2.8. *The KL divergence between two probabilistic grammars G_1 and G_2 is:*

$$d_{KL}(G_1,G_2) = \sum_{x \in \Sigma^*} Pr_{G_1}(x) \left(\log Pr_{G_1}(x) - \log Pr_{G_2}(x) \right) \tag{4.2}$$

Even if the KL-divergence does not respect the triangular inequality, $d_{KL}(G_1,G_2) = 0 \iff G_1 \equiv G_2$.

Definition 4.2.9. *Let G be a PCFG. The consensus string for G is the most probable string of $\mathscr{D}_G$. We denote by $\langle CS, \mathbf{PCFG}(\Sigma) \rangle$ the decision problem: is w the most probable string given G?*

4.2.3 PCP and the probabilistic grammars

Definition 4.2.10. *For each distance d_X the problem $\langle d_X, \mathscr{G} \rangle$ is the decision problem: given two grammars G and G' from $\mathscr{G}$, and any rational k, do we have $d_X(G,G') \leq k$?*

We use *Turing reduction* between decision problems and write:

$$\Pi_1 \leq_T \Pi_2$$

for problem Π_1 reduces to problem Π_2: if there exists a terminating algorithm solving Π_2 there also is one solving Π_1. If simultaneously $\Pi_1 \leq_T \Pi_2$ and $\Pi_1 \leq_T \Pi_2$, we will say that Π_1 and Π_2 are interreducible. The construction can be used for non decision problems: if only Π_1 is a decision problem, Π_1 is undecidable, and $\Pi_1 \leq_T \Pi_2$, we will say that Π_2 is *uncomputable*.

One particular well-known undecidable problem can be used as starting point for the reductions: the Post Correspondence Problem (Post, 1946), which is undecidable:

Name: PCP
Instance: A finite set F of pairs of strings $(u_i, v_i), 1 \leq i \leq n$ over an alphabet Σ.
Question: Is there a finite sequence of integers $x_1 \ldots x_t$, $t > 0$ such that $u_{x_1} u_{x_2} \ldots u_{x_t} = v_{x_1} v_{x_2} \ldots v_{x_t}$?

We give two standard constructions starting from an instance F of PCP. In both cases we use another alphabet containing one symbol $\#_i$ for each $i : 1 \leq i \leq n$. Let Ω denote this alphabet.

Construction 1: Two grammars
An instance of PCP as above is transformed into two PCFGs G_1 and G_2 as follows:
Rules of G_1: $S_1 \rightarrow u_i S_1 \#_i$ and $S_1 \rightarrow u_i \#_i$
Rules of G_2: $S_2 \rightarrow v_i S_2 \#_i$ and $S_2 \rightarrow v_i \#_i$
Each rule has probability $\frac{1}{2n}$.

Construction 2: One grammar
An instance of PCP is first transformed into two PCFGs G_1 and G_2 as above. Then a new non-terminal S_0 is introduced and we add the new rules $S_0 \rightarrow S_1$ and $S_0 \rightarrow S_2$, each with probability $\frac{1}{2}$.

The language obtained through G_0, G_1 and G_2 contains only strings x which can always be decomposed into $x = yz$ with $y \in \Sigma^*$ and $z \in \Omega^*$. We note that the number of derivation steps to generate string x is $1 + |z|$ for G_1 and G_2. For a final string x we denote this number by $\text{len}(x)$. For example $\text{len}(aababababa\#_3\#_1\#_4\#_1) = 4$.

Note that a positive instance of PCP will lead to G_1 and G_2 with a non empty intersection, and to an ambiguous G_0.

The following properties hold:

Property 1. – *G_1 and G_2 are unambiguous and deterministic.*
– *If $x \in \mathbb{L}(G_1)$, $Pr_{G_1}(x) = (\frac{1}{2n})^{\text{len}(x)}$*
– *F is a positive instance of PCP if and only if $\text{CoEm}(G_1, G_2) > 0$*
– *F is a positive instance of PCP if and only if G_0 is ambiguous. In terms of probabilities, if there exists a string x such that $Pr_{G_0}(x) = (\frac{1}{2n})^{\text{len}(x)}$*

Property 2. *Let F be an instance of PCP with n pairs. Then*

$$\text{Tac}(G_0) = \frac{1}{16n-4}$$

$$\begin{aligned}
\text{Tac}(G_0) &= \sum_{i \geq 1} \left(n^i \cdot \frac{1}{4} \cdot (\frac{1}{2n})^{2i} \right) \\
&= \frac{n}{16n^2} \cdot \sum_{i \geq 0} (\frac{1}{4n})^i \\
&= \frac{1}{16n} \cdot \frac{1}{1 - \frac{1}{4n}} \\
&= \frac{1}{16n} \cdot \frac{4n}{4n-1} \\
&= \frac{1}{16n-4}
\end{aligned}$$

4.3 Some decidability results

We report results concerning the problems related to equivalence and decision computation.

4.3.1 With the Manhattan distance

Proposition 4.3.1. *One cannot compute, given two PCFGs G_1 and G_2,*

$$d_{\mathrm{L1}}(G_1, G_2) = \sum_{x \in \Sigma^*} |Pr_{G_1}(x) - Pr_{G_2}(x)|.$$

Proof. If this were possible, then we could easily solve the *empty intersection problem* by building $\textsc{Mp}(G)$ and $\textsc{Mp}(G')$ and then checking if $d_{\mathrm{L1}}(\textsc{Mp}(G), \textsc{Mp}(G')) = 2$. □

Corollary 4.3.2. *One cannot compute, given two PCFGs G_1 and G_2, the variation distance between G_1 and G_2.*

The above follows from a straightforward application of Equation 4.1.

The same construction can be used for the Hellinger and Jenson-Shannon distances: whenever $\mathbb{L}(G_1)$ and $\mathbb{L}(G_2)$ have an empty intersection, it follows that $d_H(G_1, G_2) = 1$ and $d_{JS}(G_1, G_2) = 2$.

Summarizing, $\langle d_{\mathrm{L1}}, \mathbf{PCFG}(\Sigma)\rangle$, $\langle d_V, \mathbf{PCFG}(\Sigma)\rangle$, $\langle d_H, \mathbf{PCFG}(\Sigma)\rangle$ and $\langle d_{JS}, \mathbf{PCFG}(\Sigma)\rangle$ are undecidable.

4.3.2 With the Euclidian distance

For PFA, positive results were obtained in this case: the distance can be computed, both for PFA and HMM in polynomial time (Lyngsø et al., 1999).

Jagota et al. (2001) gave the essential elements allowing a proof that co-emission, the L2 and the Hellinger distances are uncomputable. In order to be complete, we reconstruct similar results in this section.

Proposition 4.3.3. *One cannot compute, given two* PCFGs *G_1 and G_2,*

$$\textsc{CoEm}(G_1, G_2) = \sum_{x \in \Sigma^*} Pr_{G_1}(x) \cdot Pr_{G_2}(x).$$

Proof. If this were possible, then we could easily solve the empty intersection problem, by taking G and G', building $\textsc{Mp}(G)$ and $\textsc{Mp}(G')$ and then checking if $\textsc{CoEm}(\textsc{Mp}(G), \textsc{Mp}(G')) = 0$. □

Proposition 4.3.4. *Computing the auto-coemission* AC *is at least as difficult as computing the L2 distance.*

Proof. Suppose we have an algorithm to compute the L2 distance. Then given any grammar G, we build a dummy grammar G_D which only generates, with probability 1, a single string over a different alphabet. It then follows that

$$d_{\mathrm{L2}}(G, G_D) = \sqrt{\textsc{CoEm}(G, G) - 2\textsc{CoEm}(G, G_D) + \textsc{CoEm}(G_D, G_D)},$$

and since the intersection between the support languages for G and G_D is empty, $\textsc{CoEm}(G, G_D) = 0$. On the other hand $\textsc{CoEm}(G_D, G_D) = 1$, trivially. Therefore $\textsc{CoEm}(G, G) = d_{\mathrm{L2}}(G, G_D)^2 - 1$. □

Corollary 4.3.5. *One cannot compute, given two PCFGs G_1 and G_2, the L2 distance between G_1 and G_2.*

Proof. By proposition 4.3.4 all we have to prove is that computing the auto-coemission is impossible. Let G_0 be the probabilistic context-free grammar built from an instance of PCP. Suppose we can compute $\textsc{Ac}(G_0)$. Then since (by Property 2) we can compute $\textsc{Tac}(G_0)$, one could then solve the ambiguity problem via Proposition 4.2.1. This is impossible. □

Summarizing, $\langle d_{\mathrm{L2}}, \mathbf{PCFG}(\Sigma)\rangle$ and $\langle \textsc{CoEm}, \mathbf{PCFG}(\Sigma)\rangle$ are undecidable. Furthermore $\langle \textsc{Ac}, \mathbf{PCFG}(\Sigma)\rangle$ and $\langle d_{\mathrm{L2}}, \mathbf{PCFG}(\Sigma)\rangle$ are interreducible.

4.3.3 With the KL divergence

Proposition 4.3.6. *One cannot compute, given two PCFGs G_1 and G_2, $d_{KL}(G_1, G_2)$.*

Proof. Suppose we could compute the KL divergence between two PCFGs. We should note that $d_{KL}(G_1, G_2) < \infty$ if and only if $\mathbb{L}(G_1) \subseteq \mathbb{L}(G_2)$. We would therefore be able to check if one context-free language is included in another one, which we know is impossible. □

The same proof can be used for the χ^2 distance since $d_{\chi^2}(G_1, G_2) < \infty$ if and only if $\mathbb{L}(G_1) \subseteq \mathbb{L}(G_2)$. Summarizing, $\langle d_{KL}, \mathbf{PCFG}(\Sigma)\rangle$ and $\langle d_{\chi^2}, \mathbf{PCFG}(\Sigma)\rangle$ are undecidable.

4.3.4 About the consensus string

A first result is that computing the Chebyshev distance is at least as difficult as computing the most probable string : Any PCFG G can be converted into G' with the same rules as G but using a disjoint alphabet. Now, it is clear that $d_{\mathrm{L}_\infty}(G, G') = Pr_G(CS(G))$:

Proposition 4.3.7. $\langle CS, \mathbf{PCFG}(\Sigma)\rangle$ *is decidable if there exists an algorithm computing* L_∞.
Ie, $\langle CS, \mathbf{PCFG}(\Sigma)\rangle \leq_T \langle d_{\mathrm{L}_\infty}, \mathbf{PCFG}(\Sigma)\rangle$.

Proposition 4.3.7 does not preclude that $\langle CS, \mathbf{PCFG}(\Sigma)\rangle$ may be decidable if $\langle \mathrm{L}_\infty, \mathbf{PCFG}(\Sigma)\rangle$ is not.

In fact $\langle CS, \mathbf{PCFG}(\Sigma)\rangle$ is decidable:

Lemma 4.3.8. *Let* $0 < \varepsilon < 1$. *Given any consistent PCFG G, there exists* $n \geq 0$ *such that* $Pr_G(\Sigma^{>n}) < \varepsilon$

Proof. Because $\lim_{n\to+\infty} Pr(\Sigma^{\leq n}) = 1$, there exists n such that $Pr(\Sigma^{\leq n}) \geq 1 - \varepsilon$, hence $Pr(\Sigma^{>n}) < \varepsilon$. □

Proposition 4.3.9. $\langle CS, \mathbf{PCFG}(\Sigma)\rangle$ *is decidable.*

Proof. The proof for this is the existence of an algorithm that takes as input a PCFG G and always terminates by returning the consensus string for G. Algorithm 3 does exactly this.

□

Algorithm 3 goes through all the possible strings in Σ^0, Σ^1, Σ^2..., and checks the probability that G assigns to each string. It stores the string with the highest probability value (*Current_Best*) and the highest probability value itself (*Current_Prob*). It also subtracts from 1 all the probability values encountered (*Remaining_Prob*). So, after the i^{th} loop, *Current_Best* is the most probable string in $\Sigma^{<i}$, *Current_Prob* is the probability of *Current_Best* and *Remaining_Prob* is $1 - Pr(\Sigma^{<i})$ which is equal to $Pr(\Sigma^{\geq i})$. Using Lemma 4.3.8, we can say that for any ε, $0 < \varepsilon < 1$, there exists an i such that after the i^{th} iteration, *Remaining_Prob* is smaller than ε. This means that the algorithm must halt at some point. Moreover, if the most probable string in $\Sigma^{<i}$ has probability higher than $Pr(\Sigma^{\geq i})$, then we can be sure that this is the consensus string. This means that the algorithm always returns the consensus string.

4.3.5 With the Chebyshev distance

Proposition 4.3.10. $\langle d_{\mathrm{L}_\infty}, \mathbf{PCFG}(\Sigma)\rangle$ *and* $\langle EQ, \mathbf{PCFG}(\Sigma)\rangle$ *are interreducible.*

Proof. $\langle d_{\mathrm{L}_\infty}, \mathbf{PCFG}(\Sigma)\rangle \leq_T \langle EQ, \mathbf{PCFG}(\Sigma)\rangle$.
Suppose $\langle EQ, \mathbf{PCFG}(\Sigma)\rangle$ is decidable. Then if G_1 and G_2 are equivalent, $d_{\mathrm{L}_\infty}(G_1, G_2) = 0$. If G_1 and G_2 are not equivalent, there exists a smallest string x such that $Pr_{G_1}(x) \neq Pr_{G_2}(x)$. An enumeration algorithm will find this initial string x, whose probability is p. Note that if $Pr_{G_1}(\Sigma^{>n}) < p$ and $Pr_{G_2}(\Sigma^{>n}) < p$, we can be sure that no string in $\Sigma^{>n}$ has a difference of probabilities of more than p. This allows us to adapt

Algorithm 3 to reach the length n at which we are sure that no string x longer than n can have probability more than $|Pr_{G_1}(x) - Pr_{G_2}(x)|$. The algorithm will therefore halt.

The converse ($\langle EQ, \mathscr{G} \rangle \leq_T \langle d_{\mathrm{L}_\infty}, \mathscr{G} \rangle$) is trivial since $d_{\mathrm{L}_\infty}(G_1, G_2) = 0 \Leftrightarrow G_1$ and G_2 are equivalent. □

Algorithm 3: Finding the consensus string

```
Data: a PCFG G
Result: w, the most probable string
Current_Prob = 0;
Current_Best = λ;
Remaining_Prob = 1;
n = 0;
Continue = true;
while Continue do
    if Remaining_Prob < Current_Prob then
        Continue = false;
    end
    else
        foreach w ∈ Σ^n do
            p = Pr_G(w);
            Remaining_Prob = Remaining_Prob − p;
            if p > Current_Prob then
                Current_Prob = p;
                Current_Best = w;
            end
        end
        n = n + 1;
    end
end
return Current_Best
```

4.4 Discussion

The results presented in this chapter are threefold:

– the only positive result concerns the consensus string, which is computable;

– the multiple ambiguity problem (for CFGs) is equivalent to the Chebyshev distance problem (for PCFGs), which in turn is equivalent to the equivalence problem (also for PCFGs);

– all the other results correspond to undecidable problems.

Interestingly, if we consider the Chebyshev distance problem as a decision problem, namely:

Name: Chebyshev distance-$\leq$
Instance: Two PCFGs G_1 and G_2, $\varepsilon : 0 \leq \varepsilon \leq 1$
Question: Is $d_{\mathrm{L}_\infty}(G_1, G_2) \leq \varepsilon$?

the problem is actually decidable in all cases but one: when $\varepsilon = 0$. Ideally, one would hope to be able to bound the length of the strings over which the search should be done. This is possible in the case of probabilistic finite automata where it is proved that (1) a PFA can be transformed into an equivalent λ-free

PFA $\mathscr{A}$, and, (2) the length of any string of probability at least p is upper bounded by $\frac{|\mathscr{A}|}{p^2}$, with $|\mathscr{A}|$ the size (number of states+1) of the PFA (Higuera and Oncina, 2013b).

It should be noted that if the question is: Is $d_{\mathrm{L}_\infty}(G_1, G_2) < \varepsilon$?, the problem becomes decidable.

Moreover, it would be important to obtain approximation results, ie,

Name: X-distance-approx
Instance: Two PCFGs G_1 and G_2, $\varepsilon : 0 < \varepsilon \leq 1$
Question: Compute a such that $|a - d_X(G_1, G_2)| \leq \varepsilon$?

Such results have been studied in the case of probabilistic finite state machines, for example, recently, in (Chen and Kiefer, 2014). In the case of the distances used in this work, the decidability of approximation would be ensured by Lemma 4.3.8. But the question of finding good approximations in polynomial time is clearly an important problem.

The following is an interesting problem which might give insight into the question of whether PCFG equivalence is decidable or not: *Given two CFGs G_1 and G_2, can we decide whether there exists probability assignments ϕ_1 and ϕ_2 for G_1 and G_2 respectively such that the PCFG made of G_1 with probabilities ϕ_1 is probabilistically equivalent to the PCFG made of G_2 with probabilities ϕ_2?*. Apart from being equivalent, it seems that G_1 and G_2 need to be somehow structurally similar in order for their probabilistic counterparts to be equivalent. For example, the following grammars that generate the simple language a^+ cannot be assigned probabilities such that they become probabilistically equivalent:

$S \to aS \qquad S \to a$

and

$S \to SS \qquad S \to a$

In order for both grammars to generate the string a with the same probability, the rule $S \to a$ in both grammars must be assigned the same probability (say p). By definition, the other rule in both grammars must have probability $1 - p$. But now, the string aa (and every string a^i, $i \geq 2$) will not be assigned the same probability from the two grammars, due to the ambiguity of the second grammar. Therefore, these two equivalent grammars can never be probabilistically equivalent.

On the other hand, any λ-free PCFG can be transformed into an equivalent PCFG in CNF (Abney et al., 1999). The structure of the two grammars above are somewhat different whilst grammars transformed in CNF retain general structural similarities. Also, ambiguity seems to play an important role in this problem. It seems harder to have two equivalent grammars, one ambiguous and one not, such that they can be made to be probabilistically equivalent. We think that investigation of such questions can result in better understanding of the structurally similarities between equivalent PCFGs.

5

A Restricted Subclass of PCFGs

An important component of any GI algorithm is the target class of grammars that the algorithm is capable of learning. It is crucial to clearly define this class in order to prove that a GI algorithm is capable of learning something. Any additional information about the target class of grammars can be helpful for the analysis of the learning algorithm.

Using the 'standard' class of context-free grammars (from the Chomsky hierarchy) as the target class for learning is problematic. This is because it is known that the whole class of context-free language is *not* learnable under a variety of reasonable learning models (see Section 3.1) for more details). Therefore one needs to define restricted subclasses of these grammars in order to obtain positive results. Ideally, these restrictions should be 'designed' in a way that makes their learnability possible. This is exactly what we do in this chapter: define restricted classes of (P)CFGs in such a way that the learning problem is simplified. We do this by following ideas described in (Clark and Eyraud, 2007; Yoshinaka, 2008; Clark, 2010a; Yoshinaka, 2010; Coste et al., 2012, 2014), where classes of grammars are defined in such a way that the building blocks of grammars (i.e. the non-terminals and production rules) are directly related to features of their language, where such features which can be extracted from the sample given to the learning algorithm. Therefore, the learning problem is reduced to that of finding these features and representing them as non-terminals and production rules of the induced grammar.

We start this chapter by studying the case of (P)DFA learning, which is driven by the idea of linking the building blocks of (P)DFAs (i.e. the states and transition rules) with features of their language through the *Myhill-Nerode Theorem*. This serves as a motivational example for our case.

5.1 Motivation

Deterministic finite automata are language representations that are capable of describing the class of regular languages, which is strictly contained in the class of context-free languages. Formally, a DFA is a tuple $\langle Q, \Sigma, I, F, \delta \rangle$ where Q is a finite set of states, Σ is a finite set of terminal symbols, $I \subseteq Q$ is a set of initial states, $F \subseteq Q$ is a set of final states and $\delta : Q \times \Sigma \to Q$ is a transition function (which respects the property of functions such that if $\delta(q_0, a) = q_1$ and $\delta(q_0, a) = q_2$ then q_1 must be equal to q_2). A string $a_0 \dots a_n \in \Sigma^*$ is contained in the language of a DFA $A = \langle Q, \Sigma, I, F, \delta \rangle$ iff there exists a sequence of states $q_0 \dots q_n$ such that:

- $q_0 \in I$
- $q_{i+1} = \delta(q_i, a_i)$ for $i = 0 \ldots n-1$
- $q_n \in F$

Given any language L and a string $u \in \Sigma^*$, the residual language for u w.r.t. L, denoted as $R_L(u)$, is $\{v \mid uv \in L\}$. Therefore, $R_L(u)$ is simply the set of suffixes of u in the strings of L. Clearly, if $w \in R_L(x)$ and $w \in R_L(y)$ then $R_L(x) = R_L(y)$. This means that the residual languages are in fact equivalence classes over Σ^*. So we can define the equivalence relation (known as the right invariant equivalence relation) $\sim_L$ as follows:

$$u_1 \sim_L u_2 \text{ iff } R_L(u_1) = R_L(u_2)$$

The Myhill-Nerode theorem is a very important result that:

1. Gives an exact characterization of the regular languages and
2. Shows a direct link between the residual languages of all the strings of a language L and the minimal representation of L as a DFA.

Theorem 5.1.1. ***Myhill-Nerode theorem*** *(Nerode, 1958): A language L is regular iff the number of residual languages of strings in L is finite (i.e. the set of languages $\{R_L(u) \mid u \in L\}$ is finite). Moreover, there is a one-to-one correspondence between the states of the smallest DFA generating L (smallest in terms of the number of states) and the residual languages, where the set of strings generated from one state of a minimal DFA is exactly equal to one residual language.*

It is precisely the second result in the Myhill-Nerode theorem that simplifies DFA learning from text. This is because the problem is reduced to that of clustering all the prefixes found in the text sample according to the right invariant equivalence relation. Each cluster will correspond to one DFA state and the transition function will simply follow the logical relationships between the clusters (i.e. if a cluster A contains string w and a cluster B contains string wa, $a \in \Sigma$, then $\delta(A, a) = B$). The key issue here is that these clusters are defined solely on the target language. The evidence needed to infer these clusters lies only in the strings of the target language, which is exactly what the learning algorithm is given. The learning algorithm simply needs to compare prefixes and decide whether they should or should not be placed in the same cluster based on the similarity of their suffixes. The larger the sample given, the more prefixes will have common suffixes and thus the more the evidence is reliable. Although information on the target automaton can still be helpful (for example, if we know that the smallest DFA has k states, then we would know that we can stop clustering prefixes when we obtain k clusters), the fact remains that the data in a text sample by itself contains the necessary evidence for finding the clusters and subsequently learning DFAs.

An analogous approach can be taken for learning probabilistic DFAs (PDFAs). PDFAs are simply DFAs with probability values assigned to each transition, and they generate a probabilistic language. The probability of a string generated by a PDFA is simply the product of the probabilities of the transitions used to generate the string. The probabilistic residual language for a string u w.r.t. a probabilistic language $\mathscr{D}$ is:

$$R_{\mathscr{D}}(v) = \frac{\mathscr{D}(uv)}{\mathscr{D}(u\Sigma^*)}$$

An analogous Myhill-Nerode type of theorem for PDFAs is given by (Carrasco and Oncina, 1999). This is that the states of the minimal PDFA for a probabilistic language correspond exactly to all the finite number of probabilistic residual languages, where the probabilistic language generated from each state is exactly equal to one probabilistic residual language. Therefore, PDFA learning can be reduced to clustering prefixes according to the distribution over their suffixes.

The problem with learning CFGs is that there is no Myhill-Nerode type of theorem for context-free languages in general. Unlike the case for DFAs and PDFAs, there are no direct relationship between the states of a CFG (i.e. its non-terminals) and features of the language it generates. This complicates the learning problem for CFGs since there is no direct link between language theoretic constructs observable from the data (i.e. which the learner has evidence for) and the hidden non-terminals of the grammar to be learned. A natural idea to simplify the learning problem is to define and learn restricted forms of CFGs whose non-terminals are by definition related to language features that the learner has evidence for. This is exactly what (Clark and Eyraud, 2007) do. This work was subsequently followed by other works (Yoshinaka, 2008; Clark, 2010a; Yoshinaka, 2010; Coste et al., 2012) which took the same approach and they reported simple and effective learning algorithms for a variety of restricted forms of grammars.

The restrictions on CFGs used in (Clark and Eyraud, 2007; Clark, 2010a) relate non-terminals with the equivalence classes of the relation $\equiv$ over Σ^* (described in detail in this chapter). In this chapter, we define a new restricted class of PCFGs using the same approach as (Clark, 2010a). The only two differences are that we use a probabilistic extension of the $\equiv$ relation, denoted as $\cong$, and we associate non-terminals more directly with the equivalence classes. Like (Coste et al., 2012, 2014), we define variants of the $\equiv$ and $\cong$ equivalence relations (denoted as $\equiv_{i-j}$ and $\cong_{i-j}$ respectively) which are related to the way our learning algorithm works. We also discuss properties of the class of grammars we define and highlight some open questions related to this class.

5.2 Four Different Equivalence Relations

All the equivalence relations hereunder are defined over Σ^* w.r.t. either a (non-probabilistic) language L or a probabilistic language (L,ϕ) (for the sake of simplicity, we omit reference to L or (L,ϕ) in the notation when it is clear from the contexts). The equivalence class of u w.r.t. a relation $\sim$ is denoted as $[u]_\sim$. For each equivalence relation, we give examples w.r.t. the following three context-free languages L_1, L_2, L_3 and their probabilistic counterparts (described by PCFGs) (L_1,ϕ_1), (L_2,ϕ_2), (L_3,ϕ_3):

$\boldsymbol{L_1}$: $\{a^n b^n \mid n \geq 1\}$	$\boldsymbol{L_2}$: ab^*	$\boldsymbol{L_3}$: $\{a^n b^m \mid n \geq 1\,,\, n \leq m \leq 2n\}$
$\boldsymbol{(L_1,\phi_1)}$:	$\boldsymbol{(L_2,\phi_2)}$:	$\boldsymbol{(L_3,\phi_3)}$:
$S \to aSb$ (0.4)	$S \to aB$ (0.4)	$S \to aSb$ (0.1)
$S \to ab$ (0.6)	$S \to a$ (0.6)	$S \to aSbb$ (0.2)
	$B \to bB$ (0.3)	$S \to ab$ (0.3)
	$B \to b$ (0.7)	$S \to abb$ (0.4)

5.2.1 The congruence relation $\equiv$

Consider the language L. For any substring w of L, the set of *global contexts* of w w.r.t. L, denoted $C(w)$, is $\{(l,r) \in \Sigma^* \times \Sigma^* \mid lwr \in L\}$. Two strings u and v are *syntactically congruent* with respect to L, written $u \equiv v$, if $C(u) = C(v)$. This means that a strings u and v are syntactically congruent w.r.t. language L if u and v share the exact same global contexts in L. The $\equiv$ relation is a congruence relation, which means that if $u_1 \equiv u_2$ and $v_1 \equiv v_2$ then $u_1 v_1 \equiv u_2 v_2$. The $\equiv$ relation is different from the right invariant equivalence relation defined earlier, which groups together prefixes which share the same suffixes. The relation $\equiv$ considers the relation between substrings and contexts rather than prefixes and suffixes. Clearly, $\equiv$ is a more stricter relation since if $u \equiv v$ then it follows that u and v must share the same contexts of the form (λ, r) (meaning that prefix occurrences of u and v must have the same suffixes).

Note that if L is a context-free language, then the total number of equivalence classes (i.e. the number of $[u]_{\equiv}$ for each $u \in \Sigma^*$) can potentially be infinite. In fact, this is the case for the context-free language L_1, as shown in the example below.

Examples ($\equiv$):

$\boldsymbol{L_1}$: $C(aab) = C(aaabb) = \{(a^i, b^{i+1}) \mid i \geq 0\}$
$[a]_{\equiv} = \{a\}$ $[b]_{\equiv} = \{b\}$ $[ab]_{\equiv} = \{a^i b^i \mid i \geq 1\}$
$[aab]_{\equiv} = [aaabb]_{\equiv} = \{a^{i+1} b^i \mid i \geq 1\}$
$[aaab]_{\equiv} = [aaaabb]_{\equiv} = \{a^{i+2} b^i \mid i \geq 1\}$
$\vdots$ $\vdots$ $\vdots$

$\boldsymbol{L_2}$: $C(a) = C(ab) = \{(\lambda, b^i) \mid i \geq 0\}$
$[a]_{\equiv} = [ab]_{\equiv} = ab^*$

$\boldsymbol{L_3}$: $C(ab) = \{(a^i, b^j) \mid i \geq 0\,,\, i \leq j \leq 2i+1\}$
$C(aabb) = \{(a^i, b^j) \mid i \geq 0\,,\, i \leq j \leq 2i+2\}$
$[ab]_{\equiv} = \{ab\}$ $[aabb]_{\equiv} = \{aabb\}$

5.2.2 The equivalence relation $\equiv_{i-j}$

Consider the language L. For any $i, j \geq 1$, the set of *i-j-local contexts* of a substring w of L, denoted $C_{i-j}(w)$, is $\{(l,r) \in (LC_i \times RC_j) \mid ulwrv \in L_{\#}, u, v \in \Sigma_{\#}^*\}$, where $LC_i = \Sigma^i \cup \{\#\}.\Sigma^{\leq i-1}$, $RC_j = \Sigma^j \cup \Sigma^{\leq j-1}.\{\#\}$, $L_{\#} = \{\#w\# \mid w \in L\}$ and $\Sigma_{\#} = \Sigma \cup \{\#\}$ s.t. $\# \notin \Sigma$. Note that we use the # marker to signify that a left or right context is taken up to the end of the string. Two strings u and v are *i-j-syntactically equivalent* with respect to L, written $u \equiv_{i-j} v$, if_{def} $C_{i-j}(u) = C_{i-j}(v)$. Thus, the i-j-syntactic equivalence is a weaker analogue of the syntactic congruence because it requires that strings only have local contexts in common.

Note that $\equiv_{i-j}$ is not a congruence relation. This can be shown to be true with a simple counter-example: consider the singleton language $L_0 = \{cacbc\}$. a and b are 1-1-syntactically equivalent w.r.t. L_0 because they share exactly the same 1-1-local contexts (in this case, they only have one local context which is (c,c)). Also, it is clear that $c \equiv_{1-1} c$. Thus, if $\equiv_{1-1}$ is a congruence relation then $ac \equiv_{1-1} bc$ should be true, which in fact it clearly isn't.

Unlike the case for the $\equiv$ relation, the total number of equivalence classes $[u]_{\equiv_{i-j}}$ for all $u \in L$ will always be finite. This is because the number of equivalence classes are bounded by the number of i-j-local contexts, which in turn are bounded by i, j and Σ.

Examples ($\equiv_{i-j}$):

$\boldsymbol{L_1}$: $C_{2-2}(aab) = C_{2-2}(aaabb) = \{(\#, b\#), (\#a, bb), (aa, bb)\}$
$[aab]_{\equiv_{2-2}} = [aaabb]_{\equiv_{2-2}} = \{a^{i+1} b^i \mid i \geq 1\}$

$\boldsymbol{L_2}$: $C_{5-3}(a) = C_{5-3}(ab) = \{(\#, \#), (\#, b\#), (\#, bb\#), (\#, bbb)\}$
$[a]_{\equiv_{5-3}} = [ab]_{\equiv_{5-3}} = ab^*$

$\boldsymbol{L_3}$: $C_{1-1}(ab) = C_{1-1}(aabb) = \{(\#, \#), (\#, b), (a, b)\}$
$[ab]_{\equiv_{1-1}} = [aabb]_{\equiv_{1-1}} = \{a^i b^i \mid i \geq 1\}$

5.2.3 The congruence relation $\cong$

Consider the probabilistic language (L, ϕ). For every substring w of L, we can define an *infinite discrete random variable* $\mathbf{X_w}$ whose outcomes correspond to all the possible global contexts from alphabet Σ (i.e.

$\Sigma^* \times \Sigma^*$) and whose probability mass function is

$$\mathbf{Pr}[X_w = (l,r)] = \frac{\phi(lwr)}{\mathbf{E}[w]}$$

where $\mathbf{E}[w]$ is the expected value of substring w, defined as follows:

$$\mathbf{E}[w] = \sum_{l',r' \in \Sigma^*} \phi(l'wr')$$

So, $\mathbf{E}[w]$ can be thought of as the expected number of occurrences of substring w in one string taken at random from the distribution (L,ϕ). Therefore, $\mathbf{E}[w]$ is not a probability value and can be greater than 1 (this can happen because the summation can add together more than once the probability of one particular string).

Intuitively, the probability of an outcome (l,r) from the random variable X_w can be thought of as the answer to the following question: *Given one particular occurrence of a substring w in L, what is the probability that the prefix up to w is equal to l and the suffix after w is equal to r?*. Two strings u and v are *stochastically congruent* with respect to (L,ϕ), written $u \cong v$, if_{def} X_u is equal to X_v. This is a congruence relation on Σ^*. The following is a clearer definition of $\cong$:

Definition 5.2.1. *u and v are stochastically congruent ($u \cong v$) w.r.t. a probabilistic language (L,ϕ) if_{def}*

1. *u and v are syntactically congruent ($u \equiv v$) w.r.t. L and*
2. *for all $l,r \in \Sigma^*$, $\dfrac{\phi(lur)}{\phi(lvr)} = \dfrac{\mathbf{E}[u]}{\mathbf{E}[v]}$*

Examples ($\cong$):
$(\boldsymbol{L_1,\phi_1})$: for $i \geq 0$,

$$\mathbf{Pr}[X_{aab} = (a^i, b^{i+1})] = \frac{\phi(a^i aabb^{i+1})}{\mathbf{E}[aab]} = \frac{(0.4)^{i+1}(0.6)}{0.4} = (0.4)^i(0.6)$$

$$\mathbf{Pr}[X_{aaabb} = (a^i, b^{i+1})] = \frac{\phi(a^i aaabbb^{i+1})}{\mathbf{E}[aaabb]} = \frac{(0.4)^{i+2}(0.6)}{0.16} = (0.4)^i(0.6)$$

$[aab]_\cong = [aaabb]_\cong = \{a^{i+1}b^i \mid i \geq 1\}$

$(\boldsymbol{L_2,\phi_2})$: for $i \geq 0$,

$$\mathbf{Pr}[X_a = (\lambda, b^i)] = \frac{\phi(ab^i)}{\mathbf{E}[a]}$$

where $\mathbf{E}[a] = 1$, $\phi(a) = 0.6$ and for $i \geq 1$, $\phi(ab^i) = (0.4)(0.3)^{i-1}(0.7)$.
Therefore, distribution of X_a for $i = 0,1,2,3,\ldots$ is $0.6, 0.28, 0.084, 0.0252, \ldots$

$$\mathbf{Pr}[X_{ab} = (\lambda, b^i)] = \frac{\phi(abb^i)}{\mathbf{E}[ab]} = \frac{(0.4)(0.3)^i(0.7)}{0.4} = (0.3)^i(0.7)$$

Therefore, the distribution of X_{ab} for $i = 0,1,2,3,\ldots$ is $0.7, 0.21, 0.063, 0.0189, \ldots$.
$[a]_\cong = \{a\}$ and $[ab]_\cong = ab^+$

$(\boldsymbol{L_3,\phi_3})$: for $i \geq 0$, $i \leq j \leq 2i+1$,

$$\mathbf{Pr}[X_{ab} = (a^i, b^j)] = \frac{\phi(a^i abb^j)}{\mathbf{E}[ab]} = \phi(a^i abb^j)$$

for $i \geq 0$, $i \leq j \leq 2i+2$,

$$\mathbf{Pr}[X_{aabb} = (a^i, b^j)] = \frac{\phi(a^i aabbb^j)}{\mathbf{E}[aabb]} = \frac{\phi(a^i aabbb^j)}{0.3}$$

$[ab]_{\cong} = \{ab\}$ and $[aabb]_{\cong} = \{aabb\}$

5.2.4 The equivalence relation $\cong_{i-j}$

Consider the probabilistic language (L, ϕ). For every substring w of L and for every finite $i, j \geq 1$, we can define a *finite random variable* $\mathbf{X_w^{i-j}}$ whose outcomes correspond to all the possible i-j-local contexts from alphabet Σ (i.e. $LC_i \times RC_j$) and whose probability mass function is

$$\mathbf{Pr}[X_w^{i-j} = (l, r)] = \frac{\mathbf{E}_{\#}[lwr]}{\mathbf{E}[w]}$$

where $\mathbf{E}_{\#}[lwr]$ is equal to $\mathbf{E}[lwr]$ w.r.t. the probabilistic language $(L_{\#}, \phi_{\#})$, $L_{\#} = \{\#w\# \,|\, w \in L\}$ and $\phi_{\#}(\#x\#) = \phi(x)$. Note that $\mathbf{E}[\#x\#]$ w.r.t. $(L_{\#}, \phi_{\#})$ is equal to $\mathbf{Pr}[x]$ w.r.t. (L, ϕ). Intuitively, the probability of an outcome (l, r) from the random variable X_w^{i-j} can be thought of as the answer to the following question: *Given one particular occurrence of a substring w in $L_{\#}$, what is the probability that l is equal to the first i symbols to the left of w and r is equal to the first j symbols to the right of w?*. Two strings u and v are *i-j-stochastically equivalent* with respect to (L, ϕ), written $u \cong_{i-j} v$, *if$_{def}$* X_u^{i-j} is equal to X_v^{i-j}.

Note that $\cong_{i-j}$ is not a congruence relation (for the same reason that $\equiv_{i-j}$ is not a congruence relation). The following is a clearer definition of $\cong_{i-j}$:

Definition 5.2.2. *u and v are i-j-stochastically equivalent ($u \cong_{i-j} v$) w.r.t. a probabilistic language (L, ϕ) if$_{def}$*

1. *u and v are i-j-syntactically equivalent ($u \equiv_{i-j} v$) w.r.t. L and*
2. *for all $l \in LC_i$ and for all $r \in RC_j$,* $\dfrac{\mathbf{E}_{\#}[lur]}{\mathbf{E}_{\#}[lvr]} = \dfrac{\mathbf{E}[u]}{\mathbf{E}[v]}$

Examples ($\cong_{i-j}$):

Table 5.1: (L_1, ϕ_1): $\cong_{2-2}$

	$\mathbf{E}[\,]$	$(\#, b\#)$	$(\#a, bb)$	(aa, bb)
aab	0.4	$\frac{0.24}{0.4} = 0.6$	$\frac{0.096}{0.4} = 0.24$	$\frac{0.064}{0.4} = 0.16$
$aaabb$	0.16	$\frac{0.096}{0.16} = 0.6$	$\frac{0.0384}{0.16} = 0.24$	$\frac{0.0256}{0.16} = 0.16$

5.2.5 Relationships between the four equivalence relations

For any two equivalence relations $\sim_1$ and $\sim_2$ over the same set X, $\sim_1$ is said to be *finer than or equal* to $\sim_2$ *if$_{def}$* for all $x_1, x_2 \in X$, if $x_1 \sim_1 x_2$ then $x_1 \sim_2 x_2$.

Table 5.2: (L_2, ϕ_2): $\cong_{5-3}$

	$\mathbf{E}[\,]$	$(\#,\#)$	$(\#,b\#)$	$(\#,bb\#)$	$(\#,bbb)$
a	1	$\frac{0.6}{1} = 0.6$	$\frac{0.28}{1} = 0.28$	$\frac{0.084}{1} = 0.084$	$\frac{0.036}{1} = 0.036$
ab	0.4	$\frac{0.28}{0.4} = 0.7$	$\frac{0.084}{0.4} = 0.21$	$\frac{0.0252}{0.4} = 0.063$	$\frac{0.0108}{0.4} = 0.027$

Table 5.3: (L_3, ϕ_3): $\cong_{1-1}$

	$\mathbf{E}[\,]$	$(\#,\#)$	$(\#,b)$	(a,b)
ab	1	$\frac{0.3}{1} = 0.3$	$\frac{0.4}{1} = 0.4$	$\frac{0.3}{1} = 0.3$
$aabb$	0.3	$\frac{0.03}{0.3} = 0.1$	$\frac{0.18}{0.3} = 0.6$	$\frac{0.09}{0.3} = 0.3$

Clearly, $\equiv$ is finer than or equal to $\equiv_{i-j}$ for any finite i and j (as examples, $\equiv$ is finer than $\equiv_{1-1}$ in the case of L_3 and equal in the case of L_1 and L_2). Similarly, $\cong$ is finer than or equal to $\cong_{i-j}$. Intuitively, if $i_1 \geq i_2$ and $j_1 \geq j_2$ then $\equiv_{i_1-j_1}$ is finer than or equal to $\equiv_{i_2-j_2}$. The same is true for $\cong_{i_1-j_1}$ and $\cong_{i_2-j_2}$.

An interesting comparison is between $\equiv$ and $\cong$. First of all, $\cong$ is at least as fine as $\equiv$ because if $u \cong v$ then $X_u = X_v$ and therefore the support of X_u must be equal to the support of X_v which means that u and v have the same global contexts and thus $u \equiv v$. In general, $\cong$ is finer than or equal to $\equiv$ (as examples, $\equiv$ is finer than $\cong$ in the case of L_2 and equal in the case of L_1 and L_3).

5.3 Congruential Grammars

We have shown earlier how the states of minimal DFAs and PDFAs correspond directly to features of the languages they generate (i.e. equivalence classes of the right invariant relation). This simplifies the learning problem, since these features are observable from the training data given. The problem is that there is no such correspondence for CFGs and PCFGs in general, and this makes learning them harder. This is why (Clark and Eyraud, 2007) and others define subclasses of CFGs and PCFGs which by definition have a direct link between their non-terminals and features of their language. In our case, we define a subclass of our own by slightly changing the definition of one such class, congruential CFGs (C-CFGs) (Clark, 2010a).

Definition 5.3.1. *(Clark, 2010a) A CFG generating the language L is said to be **congruential (C-CFG)** if for every non-terminal A, if $u \in L(A)$ then $L(A) \subseteq [u]_{\equiv}$ w.r.t. L*

The correspondence here is clear from the definition. The language generated by every non-terminal must be included or equal to one of the equivalence classes of $\equiv$ w.r.t. the language generated by the grammar. We define a new subclass of C-CFGs, which we name as strongly congruential (SC-CFG), that requires the language of every non-terminal to be *exactly equal* to one equivalence class of $\equiv$ (i.e. not included).

Definition 5.3.2. *A CFG generating the language L is said to be **strongly congruential (SC-CFG)** if for every non-terminal A, if $u \in L(A)$ then $L(A) = [u]_{\equiv}$ w.r.t. L*

As done for NTS grammars in (Clark, 2006), we can define congruential PCFGs (C-PCFGs) by requiring that non-terminals generate languages which are included in the stochastic congruence relation $\cong$ (instead of $\equiv$) as follows:

Definition 5.3.3. *A PCFG generating the probabilistic language (L, ϕ) is said to be **congruential (C-PCFG)** if for every non-terminal A, if $u \in L(A)$ then $L(A) \subseteq [u]_{\cong}$ w.r.t. (L, ϕ)*

Similarly, we can restrict this definition to obtain strongly congruential PCFG (SC-PCFGs) as follows:

Definition 5.3.4. *A PCFG generating the probabilistic language (L,ϕ) is said to be **strongly congruential (SC-PCFG)** if for every non-terminal A, if $u \in L(A)$ then $L(A) = [u]_{\cong}$ w.r.t. (L,ϕ)*

Apart from C-CFGs (Clark, 2010a), all other classes of grammars are defined by us as extensions of C-CFGs. SC-CFGs are themselves C-CFGs with the added restriction that strings generated by a non-terminal must be exactly equal (rather than simply included) in an equivalence class of $\equiv$. C-PCFGs are the probabilistic extension of C-CFGs, where the strings generated by every non-terminal are included in an equivalence class w.r.t. $\cong$ instead of $\equiv$. SC-PCFGs are the probabilistic extension of SC-CFGs.

First of all, it is clear that all the classes of grammars defined above satisfy the following general property:

Property 3. *For any non-terminal A, if $A \stackrel{*}{\Rightarrow} u$ and $A \stackrel{*}{\Rightarrow} v$ then $[u]$ must be equal $[v]$. The equivalence classes $[u]$ and $[v]$ are on $\equiv$ for (S)C-CFGs and on $\cong$ for (S)C-PCFGs.*

This means that any two substrings dominated by the same non-terminal must by definition by substitutable. This property is neither too restrictive or too weak. Any regular language can be described by a CFG which satisfies this property. On the other hand, there exists CFLs which cannot be described by a CFG satisfying this property. A case in point are CFLs which are the union of an infinite number of equivalence classes, such as the language $\{a^n b^m \mid n \geq m \geq 1\}$.

Following this example, this is another property which all these classes of grammars clearly satisfy:

Property 4. *The language (resp. probabilistic language) L (resp. (L,ϕ)) generated by the grammar is exactly equal to some finite union of equivalence classes of $\equiv$ (resp. $\cong$) over Σ^*. CFGs which satisfy this property are called congruential in (Boasson and Sénizergues, 1985) (Thus, we follow the definition of congruential grammars given by (Clark, 2010a) rather than this definition given by (Boasson and Sénizergues, 1985)).*

Property 3 is strictly stronger than Property 4. It is stronger because:

- If no non-terminal can generate strings from different equivalence classes, and
- by definition there should be only a finite number of starting non-terminals, and
- these starting non-terminals must generate the whole language, and
- if a string u is in the language, then all the strings in $[u]$ must also be in the language, therefore
- the whole language must be made up of some finite union of equivalence classes.

To show that Property 3 is strictly stronger than Property 4 we just need a counter-example. A trivial one is the following: $S \rightarrow aSb|ab|abb$. The language is a finite union of the equivalence classes $[ab] = \{ab\}$, $[abb] = \{abb\}$ and $[aabb] = \{a^n b^n \mid n \geq 2\}$. However, non-terminal S generates strings which are contained in different equivalence classes. That said, this does not mean that the languages generated by grammars satisfying Property 3 are strictly included in the set of languages generated by grammars satisfying Property 4.

5.3.1 C-CFGs and SC-CFGs

C-CFGs cannot describe every context-free language. As an example, take the language $\{a^n b^m | m \geq n \geq 1\}$. The strings in this language are spread out into an infinite number of different equivalence classes of $\equiv$, namely: $[ab] = \{a^n b^m | n,m \geq 1\,, m = n\}$, $[abb] = \{a^n b^m | n,m \geq 1\,, m = n+1\}$, $[abbb] = \{a^n b^m | n,m \geq 1\,, m = n+2\}$ *etc.* . We cannot have an infinite number of starting non-terminals that generate this language and thus it cannot be described using a C-CFG. However, it is known that C-CFGs are capable of describing all

Language	Grammar	Equivalence Classes
$\{a^n b^n \mid n \geq 1\}$	$S \rightarrow aSb \mid ab$	$[ab] = L(S) = L$
Palindromes with central marker	$S \rightarrow aSa \mid bSb \mid c$	$[c] = L(S) = L$
Lukasiewicz language	$S \rightarrow aSS \mid b$	$[b] = L(S) = L$
Dyck language	$S \rightarrow SS \mid aSb \mid ab$	$[ab] = L(S) = L$
Regex	$S \rightarrow S* \mid S.S \mid (S)$ $S \rightarrow S+S \mid a \mid b$	$[a] = L(S) = L$
$\{a^n b^n c^m d^m \mid n,m \geq 1\}$	$S \rightarrow XY$ $X \rightarrow aXb \mid ab$ $Y \rightarrow cYd \mid cd$	$[abcd] = L(S) = L$ $[ab] = L(X) = \{a^n b^n \mid n \geq 1\}$ $[cd] = L(Y) = \{c^n d^n \mid n \geq 1\}$
$\{w \mid w \in \{a,b\}^+, \lvert w\rvert_a = \lvert w\rvert_b\}$	$S \rightarrow AB \mid BA$ $A \rightarrow AS \mid SA \mid a$ $B \rightarrow BS \mid SB \mid b$	$[ab] = L(S) = L$ $[a] = L(A) = \{w \mid w \in \{a,b\}^+, \lvert w\rvert_a = \lvert w\rvert_b + 1\}$ $[b] = L(B) = \{w \mid w \in \{a,b\}^+, \lvert w\rvert_a + 1 = \lvert w\rvert_b\}$
$\{w \mid w \in \{a,b\}^+, 2\lvert w\rvert_a = \lvert w\rvert_b\}$	$S \rightarrow ABB \mid BAB \mid BBA$ $A \rightarrow AS \mid SA \mid a$ $B \rightarrow BS \mid SB \mid b$	$[abb] = L(S) = L$ $[a] = L(A) = \{w \mid w \in \{a,b\}^+, \lvert w\rvert_a = \lvert w\rvert_b + 1\}$ $[b] = L(B) = \{w \mid w \in \{a,b\}^+, \lvert w\rvert_a + 1 = \lvert w\rvert_b\}$

Table 5.4: Example of Congruential CFGs, with their respective language and equivalence classes

regular languages, the substitutable (Clark and Eyraud, 2007) and k-l-substitutable context-free languages. In table 5.4, we give various examples of C-CFGs. For each example, we give the language, the C-CFG and the equivalence classes that correspond to non-terminals. Note that for each non-terminal A, if $u \in L(A)$ then $L(A) \subseteq [u]$ (which shows that the grammar is congruential). In fact, in all the examples we give, $L(A)$ is always equal to $[u]$. We will return to this point when discussing SC-CFGs.

C-CFGs are closely related to NTS (Non-Terminally Separated) grammars (Boasson and Sénizergues, 1985). A CFG is NTS if for all $A, B \in N$, $\alpha\beta\gamma \in (\Sigma \cup N)^*$ such that $A \Rightarrow^* \alpha\beta\gamma$ and $B \Rightarrow^* \beta$ then $A \Rightarrow^* \alpha B\gamma$. NTS grammars are all C-CFGs, but not vice versa. We can easily prove that all NTS grammars are C-CFGs as follows:

Proof. (Clark, 2010a) We want to show that if an NTS grammar G has a non-terminal A such that $A \stackrel{*}{\Rightarrow} u$, then $L(A) \subseteq [u]$. This means that for all $v \in L(A)$ and for all contexts (l,r) of u, the string $lvr \in L(G)$ (i.e. there exists a starting non-terminal S s.t. $S \stackrel{*}{\Rightarrow} lvr$). So, for any context (l,r) of u, clearly there is a starting non-terminal S s.t. $S \stackrel{*}{\Rightarrow} lur$. By the NTS property, we can conclude from $S \stackrel{*}{\Rightarrow} lur$ and $A \stackrel{*}{\Rightarrow} u$ that $S \stackrel{*}{\Rightarrow} lAr$. Since $A \stackrel{*}{\Rightarrow} v$, then $S \stackrel{*}{\Rightarrow} lvr$, which concludes our proof. □

The following is a trivial counter-example showing that not all C-CFGs are NTS: $S \rightarrow aA$ and $A \rightarrow a$. This is clearly a C-CFG but not an NTS grammar because S does not derive AA. Still, this clearly does not prove that languages generated by NTS grammars are strictly included in the set of languages generated by C-CFGs (in this case, the equivalent NTS grammar is simply $S \rightarrow AA$ and $A \rightarrow a$). This is still an open problem and it is conjectured than in fact NTS languages coincide with languages generated by C-CFGs (Clark, 2010a).

A C-CFG converted to CNF (using the standard method explained in Section 2.1.1) remains a C-CFG. This follows from the fact that for any C-CFG and for any $\alpha \in (N \cup \Sigma)^*$, if $\alpha \stackrel{*}{\Rightarrow} u$ and $\alpha \stackrel{*}{\Rightarrow} v$ then $u \equiv v$. This statement is by definition true when $\alpha \in (N \cup \Sigma)$ and remains true for any $\alpha \in (N \cup \Sigma)^*$ by induction using the fact that for any $[u][v] \in [uv]$. This result should not be taken as trivially true for any class of grammars. The fact that any unrestricted CFG G can be transformed into an equivalent CFG G' in CNF does not imply that a restricted CFG can be transformed to CNF whilst maintaining its restriction. A case in point are NTS grammars and we can show this using the following simple NTS grammar: $S \rightarrow AAA$ and $A \rightarrow a$.

Transformed to CNF, this grammar will be: $S \to AX$ and $X \to AA$ and $A \to a$. Since $S \overset{*}{\Rightarrow} AAA$ and $X \overset{*}{\Rightarrow} AA$ then by the NTS property we should have $S \overset{*}{\Rightarrow} XA$, but this is not the case.

To our knowledge, there is no algorithm in the literature which decides whether a given CFG is actually a C-CFG. Also, we know of no algorithm which decides whether two given C-CFGs are equivalent. Considering that these algorithms exist for NTS grammars ((Engelfriet, 1994) for deciding the NTS property in polynomial time and (Senizergues, 1985) for the equivalence problem), we think that it is possible to find such algorithms for C-CFGs. Inspite of the restriction with C-CFGs, it seems that no advantage can be taken of this in order to obtain more efficient parsing algorithms than the standard algorithms for general CFGs. This is the argument that Hogendorp (1989) makes for NTS grammars, which can be applied also for C-CFGs. The issue is that NTS grammars (and C-CFGs) can be highly ambiguous (such as the grammar $S \to SS|a$) and so finding efficient parsing algorithms can be difficult.

Speaking of ambiguity, it is interesting to note that C-CFGs can be either vertically ambiguous or horizontally ambiguous (or both).

Definition 5.3.5. *(Brabrand et al., 2007) A CFG is **vertically ambiguous** if there exists two different production rules $A \to \alpha$ and $A \to \alpha'$ (where $\alpha \neq \alpha'$) and a string w such that $A \to \alpha \overset{*}{\Rightarrow} w$ and $A \to \alpha' \overset{*}{\Rightarrow} w$.*

*A CFG is **horizontally ambiguous** if there exists a production rule $A \to \alpha$ s.t. α can be split into α_1 and α_2 (i.e. $\alpha = \alpha_1\alpha_2$) such that*

$\alpha_1 \overset{*}{\Rightarrow} u$ *and* $\alpha_1 \overset{*}{\Rightarrow} u'$, *where* $u \neq u'$ *and*
$\alpha_2 \overset{*}{\Rightarrow} v$ *and* $\alpha_2 \overset{*}{\Rightarrow} v'$, *where* $v \neq v'$ *and*
uv = *u'v'*

In other words, we have horizontal ambiguity when one production rule can generate the same string in multiple ways (by splitting it differently).

The interesting thing about vertical and horizontal ambiguity is that together they characterize grammar ambiguity, i.e. a grammar is ambiguous if and only if it is horizontally or vertically ambiguous (or both).

The Dyck grammar given as an example C-CFG earlier is horizontally ambiguous (because a string such as *ababab* can be generated from $S \to SS$ in two different manners). The example C-CFG that generates the language $|w|_a = |w|_b$ is vertically ambiguous (because a string such as *abab* can be generates either from $S \to AB$ or $S \to BA$).

Regarding SC-CFGs, it is clear that every SC-CFG is a C-CFG but not vice-versa. One trivial example of a grammar which is congruential but not strongly congruential is the following: $S \to lAr \,|\, lbr$ and $A \to a$. Since $[a] = \{a, b\}$ and $L(A) = \{a\}$ then $L(A) \subset [a]$.

However, we do not know of any language which can be described using a C-CFG and not a SC-CFG. We think that these two classes of grammars generate the same class of languages. Note that all the examples of C-CFGs given earlier are also SC-CFGs.

As a consequence of the added restriction on SC-CFGs, these grammars satisfy the following property (if they do not contain redundant non-terminals):

$$\text{if } A \overset{*}{\Rightarrow} w \text{ and } B \overset{*}{\Rightarrow} w \text{ then } A = B$$

This property implies that there are no rules with the same RHS in a SC-CFG. Due to this, it may be the case that more efficient parsing algorithms (i.e. more efficient than the standard ones on general CFGs) exist for SC-CFGs. However, this is still an open problem, like the case for C-CFGs and NTS grammars.

Unlike C-CFGs, there is no guarantee that a SC-CFG transformed in CNF remains a SC-CFG. However, this does not imply that SC-CFGs in CNF are less expressive than SC-CFGs in general. The problem is that

it is difficult to show if this is true or not. This is because the straightforward proof used for C-CFGs does not work for the case of SC-CFGs.

It is clear that there are still a lot of unanswered questions on C-CFGs and SC-CFGs. What is known on them is easy to derive, but what is still unknown may be very difficult to find. Problems in formal language theory can at first sight seem easy to solve (maybe due to the simple notation) but in reality they can be quite complex (with open problems left unanswered for 20-30 years). A particularly difficult problem can be that of finding whether two different classes of grammars $\mathscr{G}_1$ and $\mathscr{G}_2$ generate the same class of languages $\mathscr{L}$. If $\mathscr{G}_1$ is a restricted version of $\mathscr{G}_2$, then it should be very easy to show that the set of languages generated by $\mathscr{G}_1$ is contained or equal to the set of languages of $\mathscr{G}_2$. The difficulty arises in proving the other way round.

5.3.2 C-PCFGs and SC-PCFGs

The relationship between C-PCFGs and SC-PCFGs is analogous to the relationship between C-CFGs and SC-CFGs. Every SC-PCFG is a C-PCFG, there are C-PCFGs which are not SC-PCFGs and we are not aware of any probabilistic language describable using a C-PCFG and not a SC-PCFG.

In proposition 5.3.1, we give a simple sufficient condition for a PCFG to be congruential.

Proposition 5.3.1. *If a PCFG G satisfies the following property:*

$$\forall w \in \Sigma^+ \ \forall l,r \in \Sigma^* \ \text{ if } A \overset{*}{\Rightarrow} w \text{ then } Pr_G(lwr) = Pr_G(lAr) \cdot Pr_G(A \overset{*}{\Rightarrow} w)$$

then G is a C-PCFG. Note that $Pr_G(A \overset{*}{\Rightarrow} w)$ is the inside probability and $Pr_G(lAr)$ is the outside probability.

Proof. Let u and v be any two strings generated from any non-terminal A in G (i.e. $A \overset{*}{\Rightarrow} u$ and $A \overset{*}{\Rightarrow} v$). We want to show that if the property above holds, then $u \cong v$. This would mean that any pair of strings in $L(A)$ are stochastically congruent, which means that $L(A) \subseteq [u]_{\cong}$. This would mean that G is a C-PCFG.

Strings u and v are stochastically congruent if the two random variables X_u and X_v (as defined in Section 5.2.3) are equal.

$$\begin{aligned}
\mathbf{Pr}[X_u = (l,r)] &= \frac{Pr(lur)}{\mathbf{E}[u]} \\
&= \frac{Pr(lur)}{\sum_{l',r' \in \Sigma^*} Pr(l'ur')} \\
&= \frac{Pr(lAr) \cdot Pr(A \overset{*}{\Rightarrow} u)}{\sum_{l',r' \in \Sigma^*} Pr(l'Ar') \cdot Pr(A \overset{*}{\Rightarrow} u)} \text{ (by the property defined in this proposition)} \\
&= \frac{Pr(lAr)}{\sum_{l',r' \in \Sigma^*} Pr(l'Ar')} \\
\mathbf{Pr}[X_v = (l,r)] &= \frac{Pr(lvr)}{\mathbf{E}[v]} \\
&= \frac{Pr(lvr)}{\sum_{l',r' \in \Sigma^*} Pr(l'vr')} \\
&= \frac{Pr(lAr) \cdot Pr(A \overset{*}{\Rightarrow} v)}{\sum_{l',r' \in \Sigma^*} Pr(l'Ar') \cdot Pr(A \overset{*}{\Rightarrow} v)} \text{ (by the property defined in this proposition)} \\
&= \frac{Pr(lAr)}{\sum_{l',r' \in \Sigma^*} Pr(l'Ar')}
\end{aligned}$$

Therefore, X_u and X_v are equal and thus G is a C-PCFG.

□

From the example grammars given in Section 5.3.1, there are clearly some grammars which retain the congruential property in the probabilistic case when assigned probabilities (i.e. they are (S)C-PCFGs). The grammars describing the languages $\{a^n b^n \mid n \geq 1\}$ and $\{a^n b^n c^m d^m \mid n, m \geq 1\}$, the grammar for palindromes with a central marker and the Lukasiewicz grammar are all (S)C-PCFGs when assigned probabilities. However, other (S)C-CFGs, such as the Dyck grammar given in Section 5.3.1, do not satisfy the congruential property when assigned probabilities. This happens because the equivalence classes under the $\equiv$ relation (used for the definition of (S)C-CFGs) are *not* the same as the equivalence classes under the $\cong$ relation (used for the definition of (S)C-PCFGs) (note that we have already shown earlier that the $\cong$ relation can be finer than or equal to the $\equiv$ relation). Inspite of this, it might be the case that other equivalent (S)C-CFGs will in fact be (S)C-PCFGs when assigned probabilities.

Therefore, given this scenario, we can raise some interesting questions, for which we still do not have clear answers:

– What is the class of (non-probabilistic) languages $\mathscr{L}$ for which there exists a (S)C-PCFG that is able to describe *some* probability distribution over every $L \in \mathscr{L}$. In other words, if we consider the underlying non-probabilistic grammars of every possible (S)C-PCFG, what is the class of languages describable using these grammars? Is it equal to the class of languages describable using (S)C-CFGs?

– What other general properties (apart from the relatively strong condition shown in proposition 5.3.2) does a PCFG need to satisfy in order for it to be a (S)C-PCFG? Is there a necessary and sufficient condition? Here we think that a grammar must have some bounded level of ambiguity in order for it to satisfy the stochastic congruence criteria. This is because with highly ambiguous grammars (such as the Dyck grammar given as an example earlier), the probabilities of strings will be made up of summations of probabilities of parse trees which grow exponentially in the length of the strings. In such cases, the equivalence classes over $\cong$ become much more finer than the classes for $\equiv$. In some cases (e.g. with an ambiguous Dyck PCFG), the strings of the language itself might end up being spread into an infinite number of equivalence classes, which makes it impossible to have a (S)C-PCFG describing such a probabilistic language.

– What properties do PCFLs describable using (S)C-PCFG possess?

6

Distributional Learning of Congruence Classes

In the previous chapter, we defined a restricted class of PCFGs whose non-terminals correspond directly to the equivalence classes of the $\cong$ relation (since $\cong$ is a congruence relation, we shall from now on call its equivalence classes *congruence classes*). The idea behind defining this restricted class of PCFGs was to reduce the learning problem to that of clustering substrings from the sample according to the $\cong$ relation. Therefore, our goal is to find these clusters by comparing substrings and deciding which substrings should be put in the same cluster based on the similarity between the probabilistic distribution of the substring's contexts. This is analogous to what is done in PDFA learning, where prefixes (in our case substrings) are clustered based on the distribution over their suffixes (in our case contexts) (Vidal et al., 2005). The subsequent problem of relating the induced congruence classes to non-terminals will be dealt with in the next chapter.

The strategy we take to find the congruence classes is similar to the ones taken by (Clark, 2006, 2010a; Shibata and Yoshinaka, 2013). Our algorithm starts from the most specific case (i.e. every substring is in a cluster of its own) and then greedily takes merging decisions (i.e. decisions to place two substrings into the same cluster). This is a generalization process; the more substrings are merged, the bigger the language generated by the grammar built from these classes. It is easy to see that certain merges will yield a grammar that generates an infinite language (for example, if a substring a in the sample is merged with a substring ab, then for every string lar in the sample, the yielded grammar will generate an infinite number of strings $la(b)^+r$). In this chapter, we give two algorithms for finding the congruence classes:

1. The first is a theoretical algorithm which has access to an oracle that returns whether or not two given substrings are contained in the same congruence class. The algorithm tries to minimize as much as possible the number of questions asked to the oracle. We describe the algorithm in Section 6.1.1 and show in Section 6.1.2 how the algorithm logically deduces merging decisions by following the condition on congruence relations (i.e. if u is congruent to u' and v is congruent to v' then uv must be congruent to $u'v'$). In Section 6.1.3, we discuss the time and space complexity of the algorithm and analyse the number of times the oracle returns either *True* or *False*.

2. The second is a practical algorithm built as an extension of the first algorithm. It replaces the oracle with a statistical test on the sample to determine whether two substrings are congruent. A slightly different strategy is taken by this algorithm so that it is more applicable in practice. We describe and discuss this algorithm in Sections 6.2.1 and 6.2.2 respectively.

6.1 Theoretical Setting

6.1.1 Learning Algorithm

Algorithm 4 describes how the congruence classes are induced using an oracle `Orac`, which returns *True* if the given two strings are substitutable, *False* otherwise. The algorithm is designed in such a way that few calls are made to this oracle. This is because in reality, the oracle would need to be substituted with some statistical test on the given sample, which will obviously not be perfectly reliable.

Substrings are merged in the procedure `CongruenceClosure` (by merged we mean that they are placed in the same congruence class). This procedure also finds and merges other pairs of substrings that should logically be merged. These are found by repeatedly applying the rule which congruence classes should follow: if u and u' are in the same class and v and v' are also in the same class then uv and $u'v'$ should be in the same class. The algorithm for `CongruenceClosure` is described in Section 6.1.2.

Algorithm 4: Learning the congruence classes in a theoretical setting

Input: Sample S, Oracle `Orac` which is given two substrings u and v and returns *True* if $u \cong v$ and *False* otherwise

Output: The congruence classes $\mathscr{C}$ over the substrings of S

```
1  Subs ← Set of all substrings of S ;
2  C ← {{w} | w ∈ Subs} ;
3  Pairs ← {(u,v) | u,v ∈ Subs , u ≠ v , u ≺ v} ;
4  Order Pairs according to v (w.r.t. ≺) and if equal according to u (w.r.t. ≺) ;
5  foreach (u,v) ∈ Pairs do
6      if (u ≠ min_≺[u]) or (v ≠ min_≺[v]) then continue ;
7      if Orac(u,v) then
8          C ← CongruenceClosure(u,v,C) ;
9      end
10 end
11 return C ;
```

The algorithm loops through all the pairs of substrings and decides for each pair whether the substrings should be merged or not. Note that the algorithm does not backtrack from its decisions (i.e. whenever two strings are merged, they are never put back in separate classes). The pairs of substrings are ordered according to $\prec$, which can be any ordering on $\Sigma^* \times \Sigma^*$ such that for any two string pairs (u_1, v_1) and (u_2, v_2), $(u_1, v_1) \prec (u_2, v_2)$ if either $v_1 <_{llo} v_2$ or $v_1 = v_2$ and $u_1 <_{llo} u_2$. Note that $<_{llo}$ is any length lexicographic order on Σ^*.

The idea behind this ordering is to let the oracle merge the shorter substring pairs, and subsequently leave the `CongruenceClosure` procedure to do the rest of the merging which follows from the one taken by the oracle. For example, consider the language $a^n b^n$ (the same language to be used in the running example below). If the decision to merge ab with $aabb$ is left to the oracle, then the remaining merges needed are done in `CongruenceClosure` (i.e. without the need to call the oracle again). This is because since trivially $a \sim a$ and $ab \sim aabb$ then $aab \sim aaabb$ (because $[a][ab] = [a][aabb]$). Similarly, since $b \sim b$ and $ab \sim aabb$ then $abb \sim aabbb$ (because $[ab][b] = [aabb][b]$). $aabb$ will then be merged with $aaabbb$ from either $a \sim a$ and $abb \sim aabbb$ or $b \sim b$ and $aab \sim aaabb$. This process will eventually merge all the strings which need to be merged. If the oracle merged $(aab, aaabb)$ first instead of $(ab, aabb)$, we would miss out on merging $(ab, aabb)$ because this cannot be deduced from $(aab, aaabb)$ by the `CongruenceClosure` procedure. In fact, if a merge (x, y) is deduced by this procedure from a merge (u, v), then (x, y) must be

bigger than (u,v) w.r.t. $\prec$. This is precisely why we order the pairs according to $\prec$, so that we maximize the number of merges done by `CongruenceClosure` (and thus reducing the number of calls to the oracle).

In line 6, the algorithm leaves the two substrings in their current congruence class. This is done because at least one of the substrings is not the smallest string (according to $\prec$) in its own congruence class. In this case, the outcome of a previous decision (precisely the decision taken on the pair of smallest substrings in both classes) determined the state of the current substrings through `CongruenceClosure`. The condition in line 6 will be true in two instances:

1. Both strings are in the same congruence class, which means that they were previously merged.
2. The strings are in different congruence classes and at least one is not the smallest in its own class. Due to the fact that substring pairs are ordered according to $\prec$, then the smallest substrings in these classes were already considered for merging. Since they were not merged, then it follows that no other substring pairs from the two classes should be merged.

A running example **(Target Grammar:** $S \rightarrow aSb(0.4) \quad S \rightarrow ab(0.6)$ **)**

Initialization (up until line 4)

$S = \{ab, aaabbb\}$

$Subs = \{a,b,aa,ab,bb,aaa,aab,abb,bbb,aaab,aabb,abbb,aaabb,aabbb,aaabbb\}$

$\mathscr{C} = \{\{a\},\{b\},\{aa\},\{ab\},\{bb\},\{aaa\},\{aab\},\{abb\},\{bbb\},\{aaab\},\{aabb\},\{abbb\},$
$\{aaabb\},\{aabbb\},\{aaabbb\}\}$

$Pairs = [(a,b),(a,aa),(b,aa),(a,ab),(b,ab),(aa,ab),(a,bb),(b,bb),(aa,bb),$
$(a,aaa),(b,aaa)\ldots,(aaabb,aaabbb),(aabbb,aaabbb)]$

From the 1st Iteration (Pair: (a,b)) to the 48th Iteration (Pair: (aa,aabb))

Condition in line 6 is always *False*, thus the oracle is called for all these iterations. The result returned from each call is *False* and thus $\mathscr{C}$ remains unchanged.

The 49th Iteration (Pair: (ab,aabb))

Condition in line 6 is *False* and the oracle returns *True*. Thus, *ab* is merged with *aabb* and the resulting $\mathscr{C}$ after congruence closure is:
$\{\{a\},\{b\},\{aa\},\{ab,aabb,aaabbb\},\{bb\},\{aaa\},\{aab,aaabb\},\{abb,aabbb\},\{bbb\},\{aaab\},\{abbb\}\}$

From the 50th Iteration (Pair: (bb,aabb)) to the 55th Iteration (Pair: (aaab,aabb))

Condition in line 6 is *True*, thus no need to call the oracle for these iterations.

From the 56th Iteration (Pair: (a,abbb)) to the 65th Iteration (Pair: (aaab,abbb))

Condition in line 6 is *False* and the oracle always returns *False*. Thus, $\mathscr{C}$ remains unchanged.

From the 66th Iteration (Pair: (aabb,abbb)) to the last Iteration (Pair: (aabbb,aaabbb))
Condition in line 6 is *True*, thus no need to call the oracle for these iterations.

end running example

6.1.2 Congruence Closure

In the previous section, we explained what the procedure `CongruenceClosure` does, but we said nothing on how this procedure can be implemented. So in this section we explain how we did this. The problem here is to find an efficient way of computing the logically deduced merges, ideally without going through unnecessary pairs of strings.

To solve this problem, we make use a data structure similar to a *directed B-hypergraphs* to represent relationships between substrings, from which we compute merges that follow as a consequence of the congruence relation. A *directed B-hypergraph* (Gallo et al., 1993) is a tuple (V,E), where V is a set of vertices and $E \subseteq 2^V \times V$ is a set of *B-hyperedges*. So, directed B-hypergraphs are a generalization of conventional directed graphs, in which the tail of an edge can have multiple vertices. We use a similar form of graph, which is defined as a tuple (V,E), where V is a set of vertices and E is a set of edges that have exactly two vertices in the tail which are *ordered* (unlike B-hyperdges, where the vertices in the tail are unordered) and one head vertex.

Given the sample S, we can build a graph of this form, which we shall call the *initial congruence graph*, as follows:

$V = Subs(S)$
$E = \{(u,v) \rightarrow uv \,|\, u,v \in \Sigma^+, uv \in Subs(S)\}$

where $Subs(S)$ is the set of substrings of S. Therefore, an edge $(u,v) \rightarrow uv$ represents the simple fact that a string u concatenated with v gives uv. Figure 6.1 is an example of an initial congruence graph for the sample $\{ab, aabb\}$ (which is obviously the same initial congruence graph for the sample $\{aabb\}$). We use coloured edges to mark the order of the tail vertices: tail vertices with a red edge come first, tail vertices with a blue edge second. The '•' signs under the vertices are simply 'entry points' for the edges to the vertices (used for simplifying the graph's design).

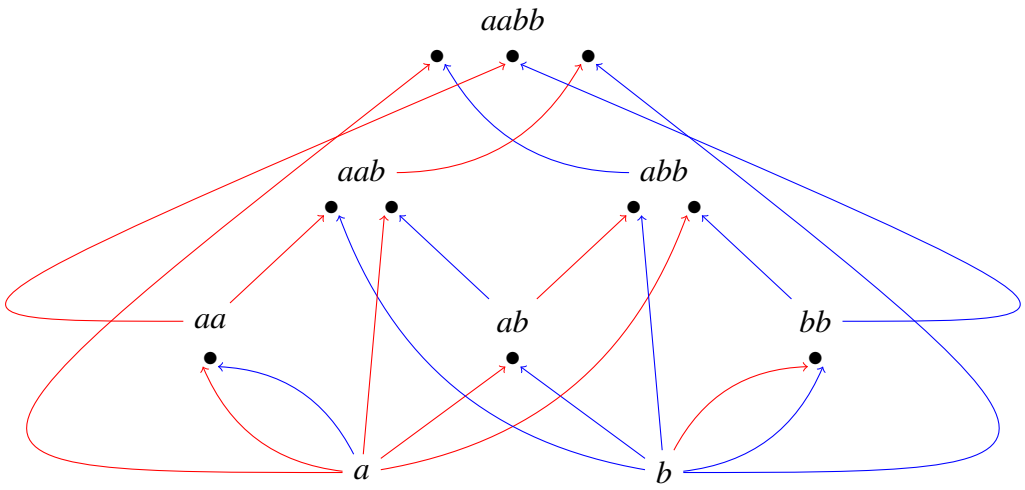


Figure 6.1: The initial congruence graph for the sample $\{ab, aabb\}$

It is clear that no two different edges in an initial congruence graph will have the same ordered tail vertices. However, if we merge two vertices from the graph, then we can potentially obtain different edges with the same ordered tail vertices. Note that by merging two vertices x and y we mean deleting any one of two

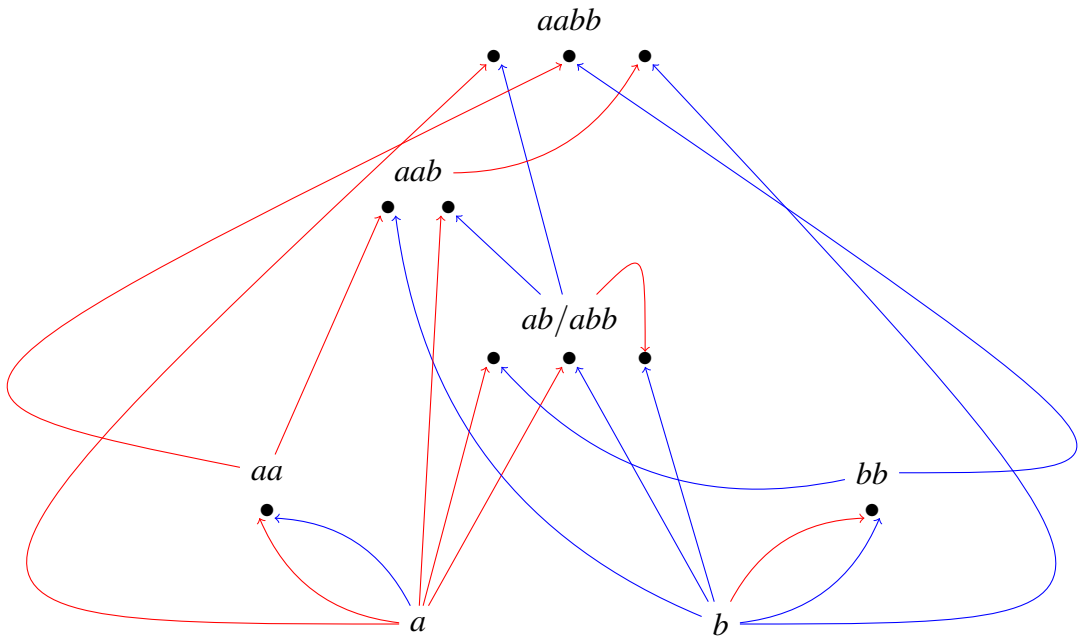


Figure 6.2: The congruence graph obtained after merging vertices *ab* and *abb* from the congruence graph in Figure 6.1

vertices and redirecting the incoming and outgoing edges from the deleted vertex to the other vertex. For example, by merging the vertices *ab* and *abb* in the initial congruence graph, we obtain the congruence graph in Figure 6.2, which contains two different edges with the same ordered tail vertices. These are the edges $(a, ab/abb) \to aab$ and $(a, ab/abb) \to aabb$. In general, two different vertices representing strings w and w' will have the same incoming edge if there is a split of string w into uv (i.e. $w = uv$) and a split of string w' into $u'v'$ (i.e. $w' = u'v'$) such that u is represented by the same vertex as u' (i.e. either $u = u'$ or u was merged with u') and v is represented by the same vertex as v' (i.e. either $v = v'$ or v was merged with v'). Clearly, this is analogous to the congruence condition where w and w' should be in the same class whenever $w = uv$, $w' = u'v'$ and $u \sim u'$ and $v \sim v'$.

Therefore, we can use congruence graphs to compute the logically deduced merges. We start from the initial congruence graph that represents the initial state where all substrings are in a congruence class of their own. When the oracle takes a merge decision on a pair (u, v), we merge the vertices u and v in the graph. The strings we will then need to merge as a consequence of the congruence relation are precisely those represented by vertices which will have the same incoming edges. After merging these vertices, more vertices might end up with the same incoming edges. Therefore, this merging process is repeated until this is not the case. This process is repeated after every merge decision taken by the oracle.

In the running example above, after merging *ab* with *aabb*, the congruence graph will contain the following: $(a, ab/aabb) \to aab$, $(a, ab/aabb) \to aaabb$ and $(ab/aabb, b) \to abb$, $(ab/aabb, b) \to aabbb$. Therefore, *aab* is merged with *aaabb* and *abb* is merged with *aabbb*. As a consequence of these merges, we will have the following: $(a, abb/aabbb) \to ab/aabb$ and $(a, abb/aabbb) \to aaabbb$, which results in $ab/aabb$ being merged with *aaabbb*.

Note that this merging procedure is similar to the merging for determinisation process in DFA learning (Higuera, 2010). In the case of DFAs, the condition is the following: if prefix u has been merged with prefix v then for any $a \in \Sigma$, prefix ua must be merged with prefix va.

6.1.3 Analysis

First of all, the whole algorithm runs in polynomial time, with a worst case running time complexity of $\mathcal{O}(n^2.k^4)$ (where n is the number of strings in the sample and k is the length of the longest string). This is because going through all the substrings of the sample requires $\mathcal{O}(n.k^2)$ time and since the algorithm has to go through pairs of substrings, then we have to square this and obtain $\mathcal{O}(n^2.k^4)$. However, in reality, the time taken will be far less than the worst case because:

- A lot of substring pairs can be easily skipped due to the condition in line 6 of the algorithm. For example, if we index all substrings according to their position in the $\prec$ ordering, then if we encounter the pair $(0,i)$ and we know that substring at index i is not the smallest in its congruence class, then we can skip to the pair $(0,i+1)$ (thus bypassing $i-1$ pairs). This will significantly reduce the running time, although in theory the worst case running time remains unchanged (because if no merges are made, then all substring pairs must be traversed).
- The more the sample contains recurring substrings, the less the total number of substrings is than $\frac{n(n+1)}{2}$ and thus the less the running time is w.r.t. n
- If the average length of strings in the sample is much less than the length k of the longest string, then the less the running time is w.r.t. k.

The space complexity is $\mathcal{O}(n.k^3)$. Although $\mathcal{O}(n^2.k^4)$ space is needed to store all the pairs of substrings, in reality we can easily generate these pairs on the fly. What we need to store is the hypergraph data structure for the congruence closure, which has $\mathcal{O}(n.k^2)$ nodes (one node for each substring) and $\mathcal{O}(n.k^3)$ edges (because the degree of each vertex is at most k).

We can also give an upper bound on the number of times the oracle returns *True* and a lower bound on the number of times the oracle returns *False*. In order to do so, we need to define a new relation, which we call the relatives relation. Two strings u and v in the same congruence class are *relatives* if either $u = v$ or there are $k \geq 2$ congruence classes $[w_1]$, $[w_2]$, ..., $[w_k]$ s.t. u and v are both in $[w_1][w_2]\dots[w_k]$ (i.e. u and v are contained in the concatenation of these congruence classes). By the condition that $[uv] \supseteq [u][v]$, we can restrict the definition w.l.o.g. to only $k = 2$. Note that the relatives relation is reflexive, symmetric and it does not need to be transitive. This is known as a *tolerance relation* (Bartol et al., 2004). We can represent the relatives relation on a particular congruence class $[w]$ as an undirected graph, where the nodes are the strings in $[w]$ and the edges are between strings that are relatives. The maximal cliques of this graph form the *blocks* of the relatives relation (Bartol et al., 2004).

Proposition 6.1.1. *Let $\{C_1, C_2, \dots C_m\}$ be the finite number of target congruence classes w.r.t. $\cong$ over all the substrings of S. Let $rb(C_i)$ be the number of relative blocks of C_i. The number of times the oracle in Algorithm 4 returns True is at most equal to the following:*

$$\sum_{i=1}^{m} \big(rb(C_i) - 1\big)$$

Proof. If two strings u and v in the same target congruence class are relatives, then we can split u and v, $u = u_1u_2$ and $v = v_1v_2$, s.t. u_1 is congruent to v_1 and u_2 is congruent to v_2. Clearly, u_1 and u_2 (resp. v_1 and v_2) are smaller than u (resp. v) w.r.t. $\prec$. Therefore, the decisions to merge u_1 with v_1 and u_2 with v_2 *precede* the decision to merge u with v. If these preceding decisions are taken correctly, then u will be merged with v as a logical consequence (i.e. u will be merged with v in `Congruence_Closure`). This is because if $u = u_1u_2$, $v = v_1v_2$, $u_1 \sim v_1$ and $u_2 \sim v_2$ then $u \sim v$. Therefore, the oracle does not need to be called on pairs of strings that are relatives. The oracle will be called and returns *True* for non-relative strings that are in the same congruence class. Merging two strings from different relative blocks will consequently

result in all the strings in the two blocks to be merged together in the `CongruenceClosure` procedure. Therefore, the merge decisions taken by the oracle are at most between the smallest strings in the relative blocks. Therefore, for each class C_i, the maximum number of times an oracle is called and returns *True* is $rb(C_i) - 1$. □

Proposition 6.1.2. *The number times the oracle returns False depends on the ordering $\prec$, but this will always be at least equal to $\frac{x(x-1)}{2}$, where $x = |\mathscr{C}|$.*

Proof. Consider the running example we gave earlier. Let's slightly change the ordering $\prec$ by placing *ab* before *aa* (note that this change does not violate the condition imposed on $\prec$ since *aa* is not a substring of *ab*). Now the number of times the oracle returns *False* decreases by 1. This is because the decision $(aa, aabb)$ will be preceded by $(ab, aabb)$, which means that by the time the algorithm reaches the iteration for $(aa, aabb)$, *aabb* will not be the smallest in its class and thus the condition in line 6 will be true. This example shows that the number of times the oracle returns *False* depends on the ordering $\prec$. However, this must be at least $\frac{x(x-1)}{2}$ ($x = |\mathscr{C}|$), because there must be at least one decision not to merge for every pair of classes in $\mathscr{C}$ so that it rules out the possibility that the classes should be merged. Note that this is not x^2 because for every pair (u, v) there is no pair (v, u). □

6.2 Practical Setting

6.2.1 Learning Algorithm

Algorithm 5 is a modified version of Algorithm 4 which is designed to be more applicable in practice. The changes made are the following:

1. The oracle is substituted with a distance function, which is used to determine whether two classes should be merged depending on the closeness of their empirical context distributions (a distance threshold d is given as a parameter to determine whether two classes are close enough).
2. Classes with the closest empirical context distributions are merged first instead of the shortest pairs of substrings. This means that the bounds on the number of merge/non-merge decisions (proved in section 6.1.3) do not apply in this practical case.
3. Only local contexts are taken into consideration (the length of the local contexts is determined by input parameter k).
4. Pairs of classes are only tested for congruence whenever there is enough evidence for them in the sample (depending on a given threshold n).
5. A stopping criterion is added so that the algorithm does not overgeneralise. Whenever the number of classes in which the language is contained falls under a pre-defined number i (given as an input parameter), the algorithm stops merging.

At the beginning, each substring (or phrase for natural language) in the sample is assigned its own congruence class (line 1). Then, pairs of *frequent* congruence classes are *merged* together depending on the *distance* between their *empirical context distribution*, which is calculated on *local contexts*. What follows is an explanation of each keyword in this statement.

Algorithm 5: Learning the congruence classes in a practical setting

Input: A multiset sample S; parameters: n, d, i; distance function `dist` on local contexts of size k
Output: The congruence classes $\mathscr{C}$ over the substrings of S

1 $Subs \longleftarrow$ Set of all substrings of S ;
2 $\mathscr{C} \longleftarrow \{\{w\} \mid w \in Subs\}$;
3 **while** *True* **do**
4 $\quad Pairs \longleftarrow \{(x,y) \mid x,y \in \mathscr{C}, x \neq y, |S|_x \geq n, |S|_y \geq n\}$;
5 $\quad$ **if** $|Pairs| = 0$ **then exitloop** ;
6 $\quad$ Order $Pairs$ based on $\texttt{dist}_k$;
7 $\quad (x,y) \longleftarrow Pairs[0]$;
8 $\quad init = \{[w]_{\mathscr{C}} \mid w \in S\}$;
9 $\quad$ **if** $\texttt{dist}_k(x,y) \geq d$ **and** $|init| \leq i$ **then exitloop** ;
10 $\quad \mathscr{C} \longleftarrow \texttt{CongruenceClosure}(x,y,\mathscr{C})$;
11 **end**
12 **return** $\mathscr{C}$;

– The *empirical context distribution* of a substring w is simply a probability distribution over all the contexts of w, where the probability for a context (l,r) is the number of occurrences of lwr in the sample divided by the number of occurrences of w (Clark, 2006; Shibata and Yoshinaka, 2013). This is extended to congruence classes by treating each substring in the class as one substring (i.e. the sum of occurrences of lw_ir, for all w_i in the class, divided by the sum of occurrences of all w_i).
– Due to the problem of sparsity with contexts (in any reasonably sized corpus of natural language, very few phrases will have more than one occurrence of the same context), only *local contexts* are considered. The local contexts of w are the pairs of first k symbols (or words for natural language) preceding and following w. The lower k is, the less sparsity is a problem, but the empirical context distribution is less accurate. For natural language corpora, k is normally set to 1 or 2.
– The substring occurrences of a congruence class x in a sample S, denoted by $|S|_x$, is the number of occurrences of every substring in class x in the sample S. A *frequent* congruence class is one whose substring occurrences in the sample add up to more than a pre-defined threshold n. Infrequent congruence classes are ignored due to their unreliable empirical context distribution. However, as more merges are made, more substrings are added to infrequent classes, thus increasing their frequency and eventually they might be considered as frequent classes.
– A *distance* function `dist` between samples of distributions over contexts is needed by the algorithm to decide which is the closest pair of congruence classes, so that they are merged together. We used L1-Distance and Pearson's chi-squared test for experiments in Chapter 8.
– The vast majority of the merges undertaken are logically deduced ones. This clearly relieves the algorithm from taking unnecessary decisions (thus reducing the chance of erroneous decisions). On the downside, one bad merge can have a disastrous ripple effect. Thus, to minimize as much as possible the chance of this happening, every merge undertaken is the best possible one at that point in time (w.r.t. the distance function used). The same idea is used in DFA learning (Lang et al., 1998).

This process is repeated until either 1) no pairs of frequent congruence classes are left to merge (line 5) or 2) the smallest distance between the candidate pairs is bigger or equal to a pre-defined threshold d and the number of congruence classes containing strings from the sample is smaller or equal to a pre-defined threshold i (line 9).

The first condition of point 2 ensures that congruence classes which are sufficiently close to each other are merged together. The second condition of point 2 ensures that the hypothesized congruence classes are

generalized enough (i.e. to avoid undergeneralization). For natural language examples, one would expect that a considerable number of sentences are grouped into the same class because of their similar structure. Obviously, one can make use of only one of these conditions by assigning the other a parameter value which makes it trivially true from the outset (0 for d and $|Subs|$ for i).

6.2.2 Discussion

Our algorithm differs from other approaches in the literature in the following aspects:

– First of all, it does not take into consideration the preference bias (i.e. decisions taken by our learning algorithm are not driven by some bias in favour of certain types of grammars; decisions are only taken based on the patterns found in the sample). The preference bias is treated in a step of its own (in fact, it is dealt with in the subsequent step of our whole algorithm, explained in the next chapter). The advantage of taking such a strategy is that the problem is split into two independent parts: the first is to generalize from the sample by finding substitution classes and the second is to find the best grammar to be built given these classes. Each can be tackled without resorting to greedy decisions that try to solve both problems at the same time. Our strategy also makes it easier to theoretically study the whole PCFG learning problem.

– Unlike many other learning algorithms (van Zaanen, 2001; Clark, 2001; Adriaans et al., 2000), whenever two substrings u and v are merged into the same class, our algorithm does *not* substitute every occurrence of u and v in the sample with some non-terminal symbol. Apart from taking the decision that u and v are substitutable, by substituting with non-terminals one is essentially taking two other decisions at the same time, which are:

  1. The decision that substrings u and v are *constituents*.
  2. The decision that *every* occurrence of u and v in the sample (at that point) must be derivable from the *same* non-terminal.

  Regarding the first decision, in our case the choice of constituents is done separately as a way to maximize on the preference bias (i.e/ it is done in the next phase of the whole algorithm). The second decision strongly limits the type of grammars that are inferred. In our case, although any substring *cannot* be derivable from two *different* non-terminals, *not every* occurrence of the same substring in the sample needs to be either derivable or not derivable by a non-terminal.

  The advantage of substituting with non-terminals is basically that one obtains better evidence for merging (i.e. After substituting two substrings with a new non-terminal symbol, every substring in the sample will have at least the same or more counts and contexts than before) With our strategy, we still benefit from this without resorting to substitutions because counts and contexts are taken on the classes of substrings. Of course, the biggest disadvantage of our strategy is that the process takes much more time compared to substituting non-terminals (although it remains polynomial w.r.t. the size of the sample given).

– Our algorithms exploits the congruence property by deducing merges.

An interesting issue concerns the difference between the $\cong$ and $\cong_{i-j}$ relations. We want our learning algorithm to be capable of resolving the question: *are u and v stochastically congruent (i.e. is* $u \cong v$*)?*, however we can only hope for an answer to the question *are u and v i-j-stochastically equivalent (i.e. is* $u \cong_{i-j} v$*)?* because we need to use local contexts. Notwithstanding this, our algorithm will still treat its answer as pertaining to the former question, even if in actual fact it is testing for the latter question. This allows our algorithm to treat the relation it is testing as a congruence, which would entail the advantage of having a congruence closure. The advantage is that for every *Yes* answer, a lot of questions are spared from the algorithm since their answer is logically deduced (using the congruence property) and thus reduces the risk of bad decisions. Moreover, the more logically deduced answers, the more informative and less

sparse the observations from the sample will be. Although we can be faced with the problematic situation where the answers to both questions are different (*No* and *Yes* respectively, vice-versa does not apply since $\cong$ is always finer than or equal to $\cong_{i-j}$), we think that the advantage of having a congruence closure far outweighs the disadvantage with this issue.

Assuming our algorithm always takes correct merge decisions, the sample required for identification needs only to be structurally complete w.r.t. the target grammar (i.e. every production rules is used at least once in the generation of the sample). This means that, in theory, our algorithm can work with very small samples (polynomial size w.r.t. the number of rules in the target grammar). In practice, larger samples are only needed so that enough statistical evidence is available for correct merging. In reality, as we show later on in Section 8, identification of different grammars is still possible from relatively small samples.

7

Building Grammars using Minimum Satisfiability

In this chapter, we explain how our algorithms builds a PCFG from the induced classes obtained from the distributional learning algorithms described in the previous chapter. We start by explaining in Section 7.1 what this task entails and how we use a preference bias to tackle it.

7.1 Motivation

We explained in Section 5.1 how DFA and PDFA learners rely on the Myhill-Nerode theorem. Using this theorem, the learning problem is reduced to that of clustering prefixes in the sample according to the right invariant equivalence relation for DFAs and its stochastic counterpart for PDFAs. Building the smallest (P)DFA from these clusters is simply an issue of assigning each cluster its own (P)DFA state and transitions between these states are built following the logical relationships between these clusters.

In our case for context-free grammars, instead of clustering prefixes, we cluster substrings based on the similarity of their contexts (as shown in the previous chapter). The problem is that, unlike the case for (P)DFAs, we cannot directly build a PCFG consistent with the data from these clusters. We can build a large 'primitive' CFG by using *all* the inferred clusters and *all* the relationships between them as non-terminals and production rules respectively. The non-probabilistic language generated by this CFG will be the same as the target non-probabilistic language (as long as we assume that the correct clusters were induced, as we show later on in this chapter). However, we have no guarantee that the PCFG obtained by assigning probabilities using the standard maximum-likelihood estimation algorithm (described in Section 2.5) will generate a probabilistic language close to the target language.

Therefore, we are faced with the problem of choosing which clusters and what relationships between these clusters should represent the non-terminals and production rules of the induced grammar. There are many different possible ways of choosing which clusters should be represented as non-terminals and the number of possible ways of choosing the relationships between these clusters is vast. This is where we apply our *preference bias* in favour of smaller representations. We treat this problem as a combinatorial optimization problem, where we try to find the *smallest* possible number of clusters and relationships between these clusters in order for a grammar to be consistent with the given data. This follows the idea in statistical inference

that the best models are those which use as few variables as possible whilst still remaining consistent with the data.

Our approach can also be seen as a transformation step from *weak learners* into *strong learners*. Practically all CFG learning algorithms which follow a distributional learning approach (Clark and Eyraud, 2007; Yoshinaka, 2008; Clark, 2006, 2010a; Shibata and Yoshinaka, 2013) (with the sole exception of (Clark, 2013)) are weak learners, which means that they give no importance to the learned structure because any grammar found that generates the target language is good enough. In reality, all of these weak learning methods end up returning very large grammars containing all combinations of valid production rules. Assigning probabilities to such grammars is not the ideal way of learning probabilistic grammars. Learning PCFGs requires that the learned structure is somewhat close to the target structure. Therefore, a strong rather than a weak learner is needed for PCFG induction, where the learned structure is taken into consideration. By choosing clusters and their relationships, we are effectively deciding on the sort of structure the induced grammar will have.

From a natural language perspective, in this step of the algorithm we are deciding which phrases are constituents, which translates into choosing which congruence classes correspond to non-terminals. This is a not a simple task and is considered a harder than the previous step of choosing substitutable phrases (Klein, 2004). A path followed by a number of authors to tackle this problem consists in using an Ockham's razor or Minimal Description Length principle approach (Stolcke, 1994; Clark, 2001). This generally leads to choosing as best hypothesis the one which best compresses the data. Applying this principle in our case would mean that the non-terminals should be assigned in such a way that the grammar built is *the smallest possible* one (in terms of the number of non-terminals and/or production rules) consistent with the congruence classes. To our knowledge, only local greedy search is used by systems in the literature which try to follow this approach. However, in our case we reduce the problem to a well-known NP-Complete problem, *Minimum Satisfiability (MIN-SAT)*, for which we can use sophisticated solvers (Berkelaar, 2008) to take care of this problem. For small examples, these solvers are able to find an exact solution in a few seconds. Moreover, these solvers are capable of finding good approximate solutions to larger formulas containing a few million variables.

In Section 7.2, we explain what we mean by a smallest grammar. We then explain in Section 7.3 how our problem is reduced to the MIN-SAT problem. We conclude with a discussion in Section 7.4.

7.2 Smallest Grammar

There are different ways of defining the size of a grammar. Five possible ways are: `N`: Number of non-terminals; `P`: Number of production rules; `N+P`: Number of non-terminals + production rules; `RHS`: Total number of symbols in the RHS of every rule; `RHS+P`: Total number of symbols in the RHS of every rule + production rules

We denote the size of a grammar G in general as $|G|$. When necessary, we will specify how the size of the grammar is calculated with reference to the 5 options above.

A *smallest grammar* for a language L in a class of grammars $\mathscr{G}$ is any grammar $G \in \mathscr{G}$ such that $L(G) = L$ and there exists no $G' \in \mathscr{G}$ with $L(G') = L$ and $|G'| < |G|$. There can be multiple smallest (S)C-CFGs in CNF for the same language. Take as an example the language $a^n b^n$, $n > 0$. The following are two smallest (S)C-CFGs in CNF for this language with respect to all grammar size options:

$S \to AX \,|\, AB \;\; X \to SB \;\; A \to a \;\; B \to b$

$S \to YB \,|\, AB \;\; Y \to AS \;\; A \to a \;\; B \to b$

7.3 Building the Grammar

Using the congruence classes learned from the previous step, we can build a *primitive grammar* G' as follows (exactly as Clark (2010a) does):

$N = \mathscr{C}$
$\Sigma = \Sigma(S)$ (i.e. the alphabet of S)
$P = \{[w] \to [u][v] \,|\, w \in Subs(S); w = uv; |u|, |v| > 0\} \cup \{[a] \to a \,|\, a \in \Sigma\}$
$S = \{[w] \,|\, w \in S\}$

Let us take as an example the congruence classes learned in the example run given in Section 6.1.1, that is:

$$\{\{a\}, \{b\}, \{aa\}, \{ab, aabb, aaabbb\}, \{bb\}, \{aaa\}, \{aab, aaabb\}, \{abb, aabbb\}, \{bbb\}, \{aaab\}, \{abbb\}\}$$

The primitive grammar for these classes is the following:

$[ab] \to [a][b] \mid [a][abb] \mid [aa][bb] \mid [aab][b] \mid [aa][abbb] \mid [aaa][bbb] \mid [aaab][bb]$
$[aab] \to [a][ab] \mid [aa][b]$
$[abb] \to [a][bb] \mid [ab][b]$
$[aaab] \to [a][aab] \mid [aa][ab] \mid [a][aab]$
$[abbb] \to [a][bbb] \mid [ab][bb] \mid [abb][b]$
$[aaa] \to [a][aa] \mid [aa][a]$
$[bbb] \to [a][aa] \mid [aa][a]$
$[aa] \to [a][a]$
$[bb] \to [b][b]$
$[a] \to a$
$[b] \to b$

Note that each non-terminal is represented by a congruence class (denoted by the smallest string in the class). Production rules simply represent all the possible concatenations of classes (following the simple fact that for any class $[xy]$, it is always true that $[xy] \supseteq [x][y]$).

Proposition 7.3.1. *Assuming that the congruence classes obtained from the previous step are correct and where built from a structurally complete sample, then the primitive grammar G' will generate the target language.*

Proof. Since it is true in general that $[uv] \supseteq [u][v]$ and $[a] \supseteq a$ then G' cannot generate strings which are not in the target language. On the other hand, the structural completeness assumption ensures that a subset of the production rules of G' are isomorphic to the target grammar, which means that G' generates at least the target language. □

If we simply want to learn any CFG that generates the target non-stochastic language, then G' is good. However, in our case we are interested in learning the stochastic language. This means that we have to find a CFG for which a probability assignment using the standard EM algorithm for PCFGs (Lari and Young, 1990) yields a stochastic language close to the target one. Using an Occam's razor argument, a good candidate grammar would be the smallest SC-CFG in CNF generating $L(G')$. However, finding such a grammar is an NP-Complete problem. To prove this, we show that this problem is at least as difficult as the Smallest CNF Grammar Problem, which is a known NP-complete problem (Rytter, 2003; Charikar et al., 2005)

Proposition 7.3.2. *The **Smallest CNF Grammar Problem**, which is the problem of finding a smallest CFG in CNF (where the size of the grammar can be either* `RHS` *or* `RHS+P`*) that generates exactly one given string w is NP-Complete (Rytter, 2003; Charikar et al., 2005)*

Theorem 7.3.3. *Given a SC-CFGs in CNF generating L, the problem of finding a smallest grammar for L in the class of SC-CFGs in CNF is NP-Complete.*

Proof. First of all, we can easily construct a SC-CFG in CNF generating one string w. Secondly, and most importantly, any smallest CFG in CNF generating only one string must be a SC-CFG since every non-terminal must generate exactly one congruence class. This is because each non-terminal must generate only one substring of w (otherwise the grammar would not generate only w) and no two non-terminals can generate the same substring (otherwise they would be redundant and thus the grammar will not be a smallest one). Thirdly, the size of a smallest CNF grammar in general in terms of `RHS` is related to the size in terms of `P` by the formula: $RHS = 2P - |\Sigma|$ and also `P` will be exactly equal to `N` (because every non-terminal is on the LHS of a rule exactly once). Therefore, a grammar is the smallest irrespective of the method used to calculate its size (`N`, `P`, `P+N`, `RHS` or `RHS+P`). Therefore, from all this we can conclude that the Smallest CNF Grammar Problem is polynomially reducible to the problem described in this theorem. □

An alternative is to find a small grammar that is bounded in size w.r.t. the smallest possible grammar. This is in fact the approach we take.

7.3.1 Building the Formula

So, our task is to decide which non-terminals/rules are to be chosen from the primitive grammar such that they form a small grammar equivalent to the target. We start by assigning a Boolean variable to every non-terminal/rule which will be true if we decide to choose that non-terminal/rule and false otherwise. We denote non-terminal variables as $N_{[x]}$, $[x]$ being the congruence class as a non-terminal, and rule variables as $R_{[y],[z]}$, representing a CNF rule whose RHS is $[y][z]$ (note that a C-CFG cannot have two different rules with the same RHS).

We can define the following constraints on these variables:

1. Any non-terminal variable whose congruence class contains strings known to be in the target language has to be true (these will be the starting non-terminals).
2. Any non-terminal variable whose congruence class contains symbols from Σ has to be true, because we need such non-terminals to define rules of the form $A \to a$.
3. If a rule variable $R_{[u],[v]}$ is true, then it is clear that $N_{[u]}$ and $N_{[v]}$ have to be true.
4. If $N_{[w]}$ is true, then for every string $w \in [w]$, there must be at least one split of w into u and v ($w = uv$) s.t. $R_{[u],[v]}$ is true. This ensures that every non-terminal representing a congruence class $[w]$ is capable of deriving each observed string in $[w]$. This constraint is consistent with the definition of SC-CFGs.

‘

Algorithm 6 describes how we can build a formula that encodes these constraints. Line 9 in the algorithm deals with the first and second constraint mentioned above, line 6 with the third constraint and line 8 with the last constraint.

As an example, let’s consider the same congruence classes as the previous example and enumerate them as follows:

1 : $[a]$, 2 : $[b]$, 3 : $[ab, aabb, aaabbb]$, 4 : $[aa]$, 5 : $[bb]$, 6 : $[aab, aaabb]$, 7 : $[abb, aabbb]$, 8 : $[aaa]$, 9 : $[bbb]$, 10 : $[aaab]$, 11 : $[abbb]$

Then the following is the formula built. Note that the leftmost clauses encode constraint 4, whilst the rest of the clauses encode constraint 3.

Variables N_1 and N_2 are true due to constraint 2, and N_3 is true due to constraint 1.

$\neg N_3 \vee R_{1,2}$	$\neg R_{1,2} \vee N_1$	$\neg R_{1,2} \vee N_2$
$\neg N_3 \vee R_{1,7} \vee R_{4,5} \vee R_{6,2}$	$\neg R_{1,7} \vee N_1$	$\neg R_{1,7} \vee N_7$
$\neg N_3 \vee R_{1,7} \vee R_{4,11} \vee R_{8,9} \vee R_{10,5} \vee R_{6,2}$	$\neg R_{4,5} \vee N_4$	$\neg R_{4,5} \vee N_5$
$\neg N_4 \vee R_{1,1}$	$\neg R_{6,2} \vee N_6$	$\neg R_{6,2} \vee N_2$
$\neg N_5 \vee R_{2,2}$	$\neg R_{4,11} \vee N_4$	$\neg R_{4,11} \vee N_{11}$
$\neg N_6 \vee R_{1,3} \vee R_{4,2}$	$\neg R_{8,9} \vee N_8$	$\neg R_{8,9} \vee N_9$
$\neg N_6 \vee R_{1,3} \vee R_{4,7} \vee R_{8,5} \vee R_{10,2}$	$\neg R_{10,5} \vee N_{10}$	$\neg R_{10,5} \vee N_5$
$\neg N_7 \vee R_{1,5} \vee R_{3,2}$	$\neg R_{1,1} \vee N_1$	
$\neg N_7 \vee R_{1,11} \vee R_{4,9} \vee R_{6,5} \vee R_{3,2}$	$\neg R_{2,2} \vee N_2$	
$\neg N_8 \vee R_{1,4} \vee R_{4,1}$	$\neg R_{1,3} \vee N_1$	$\neg R_{1,3} \vee N_3$
$\neg N_9 \vee R_{2,5} \vee R_{5,2}$	.	.
$\neg N_{10} \vee R_{1,6} \vee R_{4,3} \vee R_{8,2}$	.	.
$\neg N_{11} \vee R_{1,9} \vee R_{3,5} \vee R_{7,2}$	.	.

Algorithm 6: Building the Formula

Input: Observation table ⟨K,D,F⟩
Output: A boolean formula, which is made up of a set of clauses

```
1  Formula ← ∅ ;
2  foreach w ∈ K; |w| > 1 do
3  |   RuleVars ← ∅ ;
4  |   foreach uv = w; |u|, |v| > 0 do
5  |   |   RuleVars ← RuleVars ∪ {R_[u],[v]} ;
6  |   |   Formula ← Formula ∪ {¬R_[u],[v] ∨ N_[u], ¬R_[u],[v] ∨ N_[v]};
7  |   end
8  |   Formula ← Formula ∪ {¬N_[w] ∨ ⋁_{r∈RuleVars} r} ;
9  |   if (w ∈ D) or ([w] ∩ Σ ≠ ∅) then Set variable N_[w] to true ;
10 end
11 return Formula ;
```

The grammar built from a solution of the formula consists simply of all the production rules whose rule variables are true and the trivial rules $[a] \to a$ for every $a \in \Sigma$. This grammar will always be in CNF. However, it might contain a number of redundant rules. A rule $A \to \alpha$ is redundant if there exists no other rule in the grammar with non-terminal A on the LHS. These rules can be removed by substituting every occurrence of A in the RHS of rules with α. Clearly, the final grammar after this process will not be in CNF. For example, from the solution for the above formula with the true rule variables being $R_{1,7}$, $R_{1,2}$ and $R_{3,2}$, the CNF grammar built will be:

$[ab] \to [a][abb] \mid [a][b]$
$[abb] \to [ab][b]$
$[a] \to a \qquad [b] \to b$

By removing redundant rules, this grammar will be transformed to: $[ab] \to a\,[ab]\,b \mid ab$

We shall call *solution grammars* all the grammars built from all the possible solutions for a formula. Note that all solution grammars generate languages which are contained or are equal to the target language because the rules of these grammars are a subset of or equal to the primitive grammar's rules. For example, the following are three grammars built from three different solutions of the example formula (the leftmost and rightmost grammars generate the target language, the middle one generates a subset of the target, $[ab]$ is the starting non-terminal for all grammars):

$[ab] \to [a][abb] \mid [a][b]$
$[abb] \to [ab][b]$
$[a] \to a \quad [b] \to b$

$[ab] \to [aaa][bbb] \mid [aa][bb] \mid [a][b]$
$[aaa] \to [a][aa] \quad [bbb] \to [b][bb]$
$[aa] \to [a][a] \quad [bb] \to [b][b]$
$[a] \to a \quad [b] \to b$

$[ab] \to [aa][abbb] \mid [aa][bb] \mid [a][b]$
$[abbb] \to [ab][bb]$
$[aa] \to [a][a] \quad [bb] \to [b][b]$
$[a] \to a \quad [b] \to b$

We can use a Min-SAT solver to find an approximate *minimal* solution to the formula. MIN-SAT is the problem of finding the minimal solution that satisfies a given boolean formula, where a minimal solution is one which has the smallest number of true variables. Marathe and Ravi (1996) give a MIN-SAT solver with an approximation ratio of 2. This means that we can find a grammar that is at most twice the size of the smallest grammar in terms of `N+P`. We can get the same result on `N` and `P` by adding negligible weights to rule and non-terminal variables respectively and solve the weighted MIN-SAT problem (Marathe and Ravi, 1996) (which also has an approximation ratio of 2). Note that weighted MIN-SAT is the problem of finding the minimal weighted solution that satisfies a given boolean formula and weights on the variables of this formula, where the minimal weighted solution is one which minimizes the summation of the weights of the true variables. We can also get the same result on `RHS` and `P+RHS` because `RHS` is always equal to $2\texttt{P} - |\Sigma|$.

However, the main problem we have is that the grammar found might not be able to generate the target language (as is the case in one of the example grammars above).

7.3.2 Generalizing the Grammar

Algorithm 7: Strengthening the Formula

Input: A string w in the target language which is not accepted by the learned grammar, the primitive grammar G', the *Formula*
Output: A strengthened *Formula*

$CYKTable \leftarrow CYK(w, G')$;
$RuleVars \leftarrow \{ R_{B,C} \mid A \to BC \in CYKTable \} \cup \{ \neg R_{B,C} \mid A \to BC \in CYKTable \}$;
Add `Clause`$(\{R_{[u],[v]} \mid w = uv;\ |u|,|v| > 0\} \cap RuleVars)$ to *Formula* ;
foreach $x \in Subs(w)$, $|x| > 2$ **do**
  foreach $uv = x;\ |u|,|v| > 0$ **do**
    if $R_{[u],[v]} \notin RuleVars$ **then continue**
    if $|u| > 1$ **then**
      Add `Clause`$(\{\neg R_{[u],[v]}\} \cup \{R_{[u_1],[u_2]} \mid u = u_1u_2;\ |u_1|,|u_2| > 0\} \cap RuleVars)$ to *Formula* ;
    end
    if $|v| > 1$ **then**
      Add `Clause`$(\{\neg R_{[u],[v]}\} \cup \{R_{[v_1],[v_2]} \mid v = v_1v_2;\ |v_1|,|v_2| > 0\} \cap RuleVars)$ to *Formula* ;
    end
  end
end
return *Formula* ;

Algorithm 7 is an attempt to partially solve the problem with the previous step. If we happen to find (through more sampling for example) a string w which is in the target language and not in the learned language, we

can use this string to strengthen the boolean formula in such a way that the solution which yielded the incorrect learned grammar would be removed from the space of all possible solutions. In fact, any solution which yields a grammar that does not generate w will also be removed from the solution space.

The idea behind Algorithm 7 is to add additional clauses to the formula so that all solution grammars would have at least one parse tree for w. Since the primitive grammar G' has all the possible parse trees for w, the algorithm simply CYK parses w with G' and encodes the information in the CYK table as clauses.

One clause is added which simply says that at least one production rule that generates the whole string w must be true (line 3 of the algorithm). The rest of the added clauses follow the constraint that if a production rule $[uv] \rightarrow [u][v]$ (represented by the rule variable $R_{[u],[v]}$) from the CYK table is chosen, then for every u_1, u_2 s.t. $u_1u_2 = u$ ($|u_1|, |u_2| > 0$), at least one rule $[u] \rightarrow [u_1][u_2]$ (represented by the rule variable $R_{[u_1],[u_2]}$) from the CYK table must be chosen. The same applies for v. These constraints are added to the formula in lines 8 and 11 in the algorithm.

Note that we make use of the function `Clause` in Algorithm 7 which simply takes a finite set $\{x_1, x_2 \ldots x_n\}$ of literals (i.e. positive or negated variables) and returns a clause $x_1 \vee x_2 \vee \ldots \vee x_n$. Also note that *RuleVars* is simply the set of all positive and negated rule variables for all the production rules in the CYK table.

Algorithm 7 takes polynomial time in the length of w. Note that no new variables are introduced in the formula. Moreover, the number of possible clauses that can be added is finite because everything is bounded by the total number of rules in the primitive grammar. In case all possible clauses are added, the solution grammars will be equivalent to the primitive grammar and thus they will all generate the target language. This means that we only need to call Algorithm 7 on some finite set of strings until every solution grammar is correct. However, the big drawback is that we cannot polynomially bound this set of strings w.r.t. the size of the target grammar or the length of the strings themselves.

As an example run, we take the following grammar as the one built from the solution returned by the MIN-SAT solver:

$$[ab] \rightarrow [aaa][bbb] \mid [aa][bb] \mid [a][b]$$
$$[aaa] \rightarrow [a][aa] \quad [bbb] \rightarrow [b][bb]$$
$$[aa] \rightarrow [a][a] \quad [bb] \rightarrow [b][b]$$
$$[a] \rightarrow a \quad [b] \rightarrow b$$

This grammar does not generate the target language $\{a^n b^n \mid n \geq 1\}$. In fact, it only generates the given sample $\{ab, aaabbb\}$ and the string $aabb$. So, it does not generate strings $a^i b^i$ for $i \geq 4$.

Let's say that Algorithm 7 is given the string $aaaabbbb$. The first thing the algorithm does is to CYK parse this string with the primitive grammar. The first clause added to the formula (from line 3 in the algorithm) is the following: $R_{[a],[aab]} \vee R_{[aab],[b]} \vee R_{[aa],[abbb]} \vee R_{[aaab],[bb]}$. These variables represent all the possible first rules used by the primitive grammar in order to generate $aaaabbbb$ (which can be read from the CYK table). By adding this clause alone, it is already clear that the incorrect grammar above will not be part of the solution grammars. This is because neither one of the production rules represented by the variables in this clause are present in the incorrect grammar (and now at least one must be present in any solution grammar). Note that the rule variables $R_{[aa],[bb]}$ and $R_{[aaa],[bbb]}$ are not included because the rules $[ab] \rightarrow [aa][bb]$ and $[ab] \rightarrow [aaa][bbb]$ cannot be used as the first rules to generate $aaaabbbb$ (since they can only generate $aabb$ and $aaabbb$ respectively). The rest of the algorithm proceeds by adding more clauses (in line 8 and 11), many of which in this case will already be present in the formula. The resulting formula after this process will be more restrictive, and the space of solution grammars will be smaller. In fact, for this example, all the solution grammars will be equivalent to the target grammar after just one iteration.

7.4 Discussion

– The need to define a new subclass of CFGs which is syntactically slightly more restrictive than the congruential grammars stems from the way we build the formula. It is only natural to impose the condition that if we choose to represent an observed congruence class $\{w_1, w_2, \ldots w_n\}$ by a non-terminal, then this non-terminal has to be able to generate all the strings $w_1, w_2, \ldots w_n$. The issue is that not all solution grammars are strongly congruential, although we can easily show that they are all congruential. Whether the learned grammar is congruential or strongly congruential, we can still say that it will be bounded w.r.t. the size of the smallest possible congruential or strongly congruential grammar in CNF (assuming that the congruence classes are correct).

– It seems reasonable to extend the idea of the formula to other classes of grammars which have direct relationships between their non-terminals and observed features of the underlying language (Yoshinaka and Clark, 2010; Kasprzik and Yoshinaka, 2011).

– From a more practical point of view, we might have some a priori biases towards certain forms of grammars (apart from smaller grammars). These may be encoded in the formula by giving weights to the variables and finding the minimal weighted solution. For example, a bias can be added in favour of non-terminals representing congruence classes which, according to constituency tests (like the Mutual Information criterion in (Clark, 2001)), are more likely to contain substrings that are constituents.

– The size of the formula becomes a problem when dealing with large corpora, especially when long sample strings are given. Although modern solvers can handle formulas with a few million variables, our formulas will be much larger then this for corpora containing a few million sample strings. One solution for this problem is to minimize the formula by removing non-terminal and rule variables that most probably will not be true in the smallest solution. By doing so, we need to be ensure that the formula remains solvable and that relatively small grammars will still be included in the space of solution grammars. To our advantage, there is the fact that in general the number of true variables in the smallest solution will be far much smaller then the total number of variables. This means that the risk of removing 'good' variables will in general be small.

8

Experiments

In this section, we explain and show the results obtained from experiments we performed on our system. We tested on both artificial data (see Section 8.1) and natural language corpora (see Section 8.2).

Results on artificial data were much easier to analyse and gave us an initial idea of how our algorithm performed. Testing on artificial data is an important first step in order to check how the algorithm performs in very specific tasks. The artificial grammars on which we tested our system were designed to cover various context-free features. Also, the artificial natural language grammars we used as test cases generated natural language sentence that exhibit varying linguistic features.

The results we obtained from artificial data were encouraging. Our algorithm clearly outperforms ADIOS (Solan et al., 2005) in language modelling, which is one of the state-of-the-art GI systems for this task (Waterfall et al., 2010). Results on unsupervised parsing are comparable to those obtained by ABL (van Zaanen, 2001). However, we did not obtain positive results on natural language corpora.

8.1 Experiments on Artificial Data

We tested our system on 11 typical context-free languages and 9 artificial natural language grammars taken from 4 different sources (Stolcke, 1994; Langley and Stromsten, 2000; Adriaans et al., 2000; Solan et al., 2005). The 11 CFLs include 7 described by unambiguous grammars:

- **UC1**: $a^n b^n$
- **UC2**: $a^n b^n c^m d^m$
- **UC3**: $a^n b^m \, n \geq m$
- **UC4**: $a^p b^q$, $p \neq q$
- **UC5**: Palindromes over alphabet $\{a,b\}$ with a central marker
- **UC6**: Palindromes over alphabet $\{a,b\}$ without a central marker
- **UC7**: Lukasiewicz language ($S \rightarrow aSS|b$)

and 4 described by ambiguous grammars (which do not generate inherently ambiguous languages):

- **AC1**: $|w|_a = |w|_b$
- **AC2**: $2|w|_a = |w|_b$

– **AC3**: Dyck language
– **AC4**: Regular expressions.

The 9 artificial natural language grammars are:

– **NL1**: Grammar 'a', Table 2 in (Langley and Stromsten, 2000)
– **NL2**: Grammar 'b', Table 2 in (Langley and Stromsten, 2000)
– **NL3**: Lexical categories and constituency, pg 96 in (Stolcke, 1994)
– **NL4**: Recursive embedding of constituents, pg 97 in (Stolcke, 1994)
– **NL5**: Agreement, pg 98 in (Stolcke, 1994)
– **NL6**: Singular/plural NPs and number agreement, pg 99 in (Stolcke, 1994)
– **NL7**: Experiment 3.1 grammar in (Adriaans et al., 2000)
– **NL8**: Grammar in Table 10 (Adriaans et al., 2000)
– **NL9**: TA1 grammar in (Solan et al., 2005).

Sizes of the alphabet, number of non-terminals and production rules of these grammars are given in Table 8.1.

We tried to be representative in the choice of the 11 typical context-free languages. We included one of the simplest non-regular context-free languages $a^n b^n$ which describes a simple nested structure. The language $a^n b^n c^m d^m$ forms a simple concatenation of such structures. The Dyck language is a more general language that describes all possible nested structures. The language $|w|_a = |w|_b$ has to be described by a grammar that is capable of making one-to-one correspondence between symbols in a string. This is extended to two-to-one correspondence for the language $2|w|_a = |w|_b$. The palindrome languages (with and without a central marker) require a sort of 'mirroring' operation. A sort of counting is required for the Lukasiewicz language and the language $a^p b^q, p \neq q$. The language of regular expressions combines some features already found in other languages. In summary, we tried to include languages whose grammars exhibit varying 'features' that can be described using CFGs (and not finite state machines).

With regards to the artificial natural language grammars, we included relatively simple grammars such as **NL1** and **NL2**, used for evaluation by Langley and Stromsten (2000). **NL1** generates declarative sentences with arbitrarily long strings of adjectives (e.g. *'the dog ate'*, *'the big dog ate'*, *'the big big dog ate'*, etc.). It also includes transitive verbs like *'saw'* and intransitive verbs like *'ate'*. **NL2** generates declarative sentences with arbitrarily embedded relative clauses (e.g. *'the dog heard the cat'*, *'a dog that heard a cat saw the mouse'*, *'a dog that heard a cat that saw a mouse saw a cat'*, etc.). Slightly more complex grammars are those we took from (Stolcke, 1994). Apart from transitive and intransitive verbs, **NL3** generates predications involving prepositional phrases (e.g. 'the circle covers a square, a square is above the triangle, etc.). **NL4** is an extension of **NL3** with topicalized prepositional phrases (e.g. *'the circle above the triangle bounces'*) and prepositional phrases embedded in noun phrases (e.g. *'below a square , the triangle above the square below the square bounces'*). **NL5** is an extension of **NL3** with simple NP-internal agreement (e.g. *'a circle bounces'* vs. *'an octagon bounces'*). Finally, **NL6** is yet another extension of **NL3** where plural noun phrases are added (e.g. *'the circle is below the square'* vs. *'the circles are below the squares'*). From (Adriaans et al., 2000), we used the two grammars **NL7** and **NL8**. The former is a very simple grammar generating a finite language whilst the latter is a more complicated grammar with a larger alphabet. What **NL8** contains that other grammars do not are two kinds of noun phrases that differentiate between two types of nouns (human and non-human entities). Therefore, **NL8** generates sentences like *'John thinks that the car is small'* and *'the car looks small'* and not *'the car thinks that John is small'*. The largest and most complicated grammar is **NL9** (Solan et al., 2005), which captures structure-sensitive aspects of syntax (such as tough movement).

Ex.	$\|\Sigma\|$	$\|N\|$	$\|P\|$
UC1	2	3	4
UC2	4	7	9
UC3	2	3	5
UC4	2	5	9
UC5	2	3	5
UC6	2	3	8
UC7	2	2	3
AC1	2	4	9
AC2	2	5	11
AC3	2	3	5
AC4	7	8	13
NL1	9	8	15
NL2	8	8	13
NL3	12	10	18
NL4	13	11	22
NL5	16	12	23
NL6	19	17	32
NL7	12	3	9
NL8	30	10	35
NL9	50	45	81

Table 8.1: Size of the alphabet, number of non-terminals and productions rules of the grammars.

8.1.1 Evaluation Metrics

To evaluate the quality of the learned grammars, we need to measure two things:

1. How good the learned grammars are in assigning the correct structures (parse trees) to strings.
2. How good the learned grammars are in predicting the correct sentence probabilities (i.e. how close the learned probability distribution is to the target one).

These require two separate metrics. One metric has to compare parse trees assigned by the learned grammars with those assigned by the target grammar. The other has to compare probability distributions.

We already discussed the problem of comparing parse trees in Section 3.2.1. The standard adopted in unsupervised parsing is to compare unlabelled parse tree (i.e. the non-terminal labels are ignored) using precision and recall metrics. In our case, we follow suggestions given by van Zaanen and Geertzen (2008) and use micro-precision and micro-recall over all the non-trivial brackets. We take the harmonic mean of these two values to obtain the Unlabelled brackets F_1 score (UF_1). Note however that other systems may calculate this precision and recall metric in different manners (using macro-precision/recall and/or counting trivial brackets).

The learned distribution can be evaluated using perplexity (when the target distribution is not known) or some similarity metric between distributions (when the target distribution is known). In our case, the target distribution is known. We chose relative entropy [1] as a good measure of distance between distributions.

Our UF_1 results over test sets of one thousand strings were compared to results obtained by ABL (van Zaanen, 2001), which is a system whose primary aim is that of finding good parse trees (rather than identifying

1. The relative entropy (or Kullback-Leibler divergence) between a target distribution D and a hypothesized distribution D' is defined as $\sum_{w\in\Sigma^*} ln\left(\frac{D(w)}{D'(w)}\right)D(w)$. Add-one smoothing is used to solve the problem of zero probabilities.

the target language). Although ABL does not obtain state-of-the-art results on natural language corpora, it proved to be the best system (for which an implementation is readily available) for unsupervised parsing of sentences generated by artificial grammars. Results are shown in Table 8.2.

We calculated the relative entropy on a test set of one million strings generated from the target grammar. We compared our results with ADIOS (Solan et al., 2005), a system which obtains competitive results on language modelling (Waterfall et al., 2010) and whose primary aim is of correctly identifying the target language (rather than finding good parse trees). Results are also shown in Table 8.2.

8.1.2 Results

For the tests in the first section of Table 8.2 (i.e. above the first line), our algorithm was capable of exactly identifying the structure of the target grammar (after trivial rules with probability 1 in the learned grammar where removed and their RHS replaced in other rules). These grammars are shown in Table 8.3. Notwithstanding this positive result, the bracketing results for these tests did not always yield perfect scores. This happened whenever the target grammar was ambiguous, in which case the most probable parse trees of the target and learned grammar can be different, thus leading to incorrect bracketing.

For the tests in the second section of Table 8.2 (i.e. between the two lines), our algorithm was capable of exactly identifying the target language (but not the grammar). It is interesting to note that in all of these cases, the induced grammar was slightly smaller than the target one. The learned grammar always managed to describe the same language using one or two non-terminals and/or production rules less than the target grammar.

For the remaining tests, our algorithm did not identify the target language. In fact, it always overgeneralised. The 3 typical CFLs UC3, UC4 and UC6 are not identified because they are not contained in our subclass of CFLs. Inspite of this, the relative entropy results obtained are still relatively good. Overall, it is fair to say that the results obtained by our system, for both language modelling and unsupervised parsing on artificial data, are competitive with the results obtained by other methods.

		Relative Entropy		UF_1	
Ex.	$\|S\|$	COMINO	ADIOS	COMINO	ABL
UC1	10	**0.029**	1.876	**100**	**100**
UC2	50	**0.0**	1.799	**100**	**100**
UC5	10	**0.111**	7.706	**100**	**100**
UC7	10	**0.014**	1.257	**100**	27.86
AC1	50	**0.014**	4.526	**52.36**	35.51
AC2	50	**0.098**	6.139	**46.95**	14.25
AC3	50	**0.057**	1.934	**99.74**	47.48
AC4	100	**0.124**	1.727	**83.63**	14.58
NL7	100	**0.0**	0.124	**100**	**100**
NL1	100	**0.202**	1.646	**24.08**	**24.38**
NL2	200	**0.333**	0.963	**45.90**	**45.80**
NL3	100	**0.227**	1.491	36.34	**75.95**
NL5	100	**0.111**	1.692	**88.15**	79.16
NL6	400	**0.227**	**0.138**	36.28	**100**
UC3	100	**0.411**	0.864	61.13	**100**
UC4	100	**0.872**	2.480	42.84	**100**
UC6	100	1.449	**1.0**	**20.14**	8.36
NL4	500	**1.886**	2.918	**65.88**	52.87
NL8	1000	**1.496**	**1.531**	**57.77**	50.04
NL9	800	1.701	**1.227**	12.49	**28.53**

Table 8.2: Relative Entropy and UF_1 results of our system COMINO vs ADIOS and ABL respectively. Best results are highlighted, close results (i.e. with a difference of at most 0.1 for relative entropy and 1% for UF_1) are both highlighted

Target Language	**|Sample|**	**CNF Grammar Learned**	**Final Grammar**
UC1	10	$S \to AX \mid AB$ $X \to SB$ $A \to a \;\; B \to b$	$S \to aSb \mid ab$
UC2	50	$S \to UV$ $U \to AX \mid AB \;\; X \to UB$ $V \to CY \mid CD \;\; Y \to VD$ $A \to a \;\; B \to b$ $C \to c \;\; D \to d$	$S \to UV$ $U \to aUb \mid ab$ $V \to cVd \mid cd$
UC5	10	$S \to XA \mid YB \mid c$ $X \to AS \;\; Y \to BS$ $A \to a \;\; B \to b$	$S \to aSa \mid bSb \mid c$
UC7	10	$S \to AX \mid b$ $X \to SS \;\; A \to a$	$S \to aSS \mid b$
AC1	50	$S \to AB \mid BA$ $A \to SA \mid AS \mid a$ $B \to SB \mid BS \mid b$	$S \to AB \mid BA$ $A \to SA \mid AS \mid a$ $B \to SB \mid BS \mid b$
AC2	50	$S \to BX \mid XB$ $X \to AB \mid BA$ $B \to BS \mid SB \mid b$ $A \to AS \mid SA \mid a$	$S \to BX \mid XB$ $X \to AB \mid BA$ $B \to BS \mid SB \mid b$ $A \to AS \mid SA \mid a$
AC3	50	$S \to SS \mid XB \mid AB$ $X \to AS$ $A \to a \;\; B \to b$	$S- > SS \mid aSb \mid ab$
AC4	100	$S \to SN \mid XR \mid a \mid b$ $N \to OS \mid !$ $X \to LS$ $L \to (\;\; R \to)$ $O \to + \mid .$	$S \to SN \mid (S)$ $S \to a \mid b$ $N \to OS \mid !$ $O \to + \mid .$

Table 8.3: Grammars which our algorithm was capable of exactly identifying their structure

8.2 Natural Language Experiments

We also experimented on natural language corpora. For unsupervised parsing, we tested our system on the WSJ10 corpus, using POS tagged sentences as input. We changed the distance function used by the first phase of our algorithm from L1-Distance to Pearson's χ^2 test.

In a first experiment (inspired by the one done in Luque and López (2010)), we constructed the best possible SC-CFG consistent with the merges done in the first phase and assigned probabilities to this grammar using Inside-Outside. In other words, we ran the second phase of our system in a supervised fashion by using the treebank to decide which are the best congruence classes to choose as non-terminals. The CNF grammar we obtained from this experiment (COMINO-UBOUND) gives very good parsing results which outperform results from state-of-the-art systems DMV+CCM (Klein, 2004), U-DOP (Bod, 2006a), UML-DOP (Bod, 2006b) and Incremental (Seginer, 2007) as shown in Table 8.4. Moreover, the results obtained are very close to the best results one can ever hope to obtain from any CNF grammar on WSJ10 (CNF-UBOUND) (Klein, 2004). However, the grammar we obtain does not generalise enough and does not describe a good language model. In a second experiment, we ran the complete COMINO system. The grammar obtained from this experiment did not give competitive parsing results.

The first experiment shows that the merge decisions taken in the first phase do not hinder the possibility of finding a very good grammar for parsing. This means that the merge decisions taken by our system are good in general. Manual analysis on some of the merges taken confirms this. This experiment also shows that there exists a non-trivial PCFG in our restrictive class of grammars that is capable of achieving very good parsing results. This is a positive sign for the question of how adequate SC-PCFGs are for modelling natural languages. However, the real test remains that of finding SC-PCFGs that generate good bracketings *and* good language models. The second experiment shows that the second phase of our algorithm is not giving good results. This means that the smallest possible grammar might not be the best grammar for parsing. Therefore, other criteria alongside the grammar size are needed when choosing a grammar consistent with the merges.

Model	UP	UR	UF_1
State-of-the-art			
DMV+CCM	69.3	88.0	77.6
U-DOP	70.8	88.2	78.5
UML-DOP	-	-	82.9
Incremental	75.6	76.2	75.9
Upper bounds			
COMINO-UBOUND	75.8	96.9	85.1
CNF-UBOUND	78.8	100.0	88.1

Table 8.4: Parsing results on WSJ10. Note that *Incremental* is the only system listed as state-of-the-art which parses from plain text and can generate non-binary trees

8.3 Discussion

In order to improve our system, we think that our algorithm has to take a less conservative merging strategy in the first phase. Although the merges being taken are mostly correct, our analysis shows that not enough merging is being done. The problematic case is that of taking merge decisions on (the many) infrequent long phrases. Although many logically deduced merges involve infrequent phrases and also help in increasing the frequency of some long phrases, this proved to be not enough to mitigate this problem. Attempts

to overcome this problem ended up in over-generalization. As for future work, we think that clustering techniques can be used to help solve this problem.

A problem faced by the system is that, in certain cases, the statistical evidence on which merge decisions are taken does not point to the intuitively expected merges. As an example, consider the two POS sequences "DT NN" and "DT JJ NN" in the WSJ corpus. Any linguist would agree that these sequences are substitutable (in fact, they have lots of local contexts in common). However, statistical evidence points otherwise, since their context distributions are not close enough. This happens because, in certain positions of a sentence, "DT NN" is far more likely to occur than "DT JJ NN" (w.r.t. the ratio of their total frequencies) and in other positions, "DT JJ NN" occurs more than expected. The following table shows the frequencies of these two POS sequences over the whole WSJ corpus and their frequencies in contexts (#,VBD) and (IN,#) (the symbol # represents the end or beginning of a sentence):

	Totals	(#,VBD)	(IN,#)
"DT NN"	42,222	1,034	2,123
"DT JJ NN"	15,243	152	1,119
Ratios	3.16	6.80	1.90

It is clear that the ratios do not match, thus leading to context distributions which are not close enough. Thus, this shows that basic sequences such as "DT NN" and "DT JJ NN", which linguists would group into the same concept NP, are statistically derived from different sub-concepts of NP. Our algorithm is finding these sub-concepts, but it is being evaluated on concepts (such as NP) found in the treebank (created by linguists).

From the experiments we did on artificial natural language grammars, it resulted that the target grammar was always slightly bigger than the learned grammar. Although in these cases we still managed to identify the target language or have a good relative entropy result, the bracketing results were in general not good. This and our second experiment on the WSJ10 corpus show that the smallest possible grammar might not be the best grammar for bracketing. To not rely solely on finding the smallest grammar, a bias can be added in favour of congruence classes which, according to constituency tests (like the Mutual Information criterion in Clark (2001)), are more likely to contain substrings that are constituents. This can be done by giving different weights to the congruence class variables in the formula and finding the solution with the smallest sum of weights of its true variables.

The use of POS tags as input can also have its problems. Although we solve the lexical sparsity problem with POS tags, at the same time we lose a lot of information. In certain cases, one POS sequence can include raw phrases which ideally are not grouped into the same congruence class. To mitigate this problem, we can use POS tags only for rare words and subdivide or ignore POS tags for frequent words such as determinants and prepositions. This will reduce the number of raw phrases represented by POS sequences whilst keeping lexical sparsity low.

9

Conclusions

Our main contribution in this thesis is a practical PCFG learning algorithm with some proven properties and based on a principled approach. We define a new subclass of PCFGs (very similar to the one defined in Clark (2010a)) and use distributional learning and MDL-based techniques in order to learn this class of grammars. The algorithm was capable of inducing accurate yet compact artificial natural language grammars and typical context-free grammars.

A minor contribution in this thesis is a compendium of undecidability results for distances between PCFGs along with two positive results on PCFGs. Having such results can help in the process of finding learning algorithms for PCFGs.

We conclude with some final remarks, open problems and suggestions for future work:

– Defining a new class of grammars is easy. Proving properties about new classes and comparing these with other well-know classes can on the other hand be very difficult. For example, in our case, we do not know whether every language describable using a C-CFG can also be described using a SC-CFG. We can easily show that if congruence classes are equal to the union of non-terminal languages, then these non-terminals can be merged together without changing the language and the grammar becomes strongly congruential. The problematic case is when we have $L(A) \subset [w]$ and some string $w_1 \in [w]$ which is not derivable by any non-terminal, which is the case in the following simple grammar:

  $S \to lXr \,|\, lbr \qquad X \to a$

  $L(X) = \{a\}$ and $[a] = \{a, b\}$, so $L(X) \subset [a]$ and $b \in [a]$ is not dominated by any non-terminal. In this very simple case, it is easy to derive an equivalent strongly congruential grammar:

  $S \to lXr \qquad X \to a|b$

  However, we do not know whether in such cases it is always possible to build a SC-CFG. We conjecture that it is possible, but we do not know how to do this when an infinite number of strings like b exist (if this can be the case).

– A problem we encountered in the evaluation phase was that there are unclear evaluation benchmarks for unsupervised parsing and language modelling. It is unfair to compare two systems which were tested on either different corpora (WSJ, ATIS, CHILDES), different metrics (which brackets are to be taken into consideration when computing the precision and recall?) or different data as input (POS tagged vs. raw sentences as input). There should be more harmonisation in this aspect.

– We feel that a complete theoretical result based on the material from our major contribution is possible. One idea is to try to get a stronger type of MAT-learning result where the learned grammar, apart from being equivalent to the target, is also bounded in size w.r.t. the smallest possible grammar. Starting from the learner in (Clark, 2010a), which returns the primitive grammar and the congruence classes, we can find a small grammar through the formula. Using equivalence queries, we can check if this grammar is equivalent to the target. If not equivalent, we can use the returned counter-example string to strengthen the formula using a procedure similar to the one explained in section 7.3.2 until a correct small grammar is found. The problem with our procedure is that we cannot guarantee that a correct grammar is found after only a polynomial number of calls.

  This type of stronger learning result will be the first of its kind. Only recently has this kind of problem been considered in grammatical inference (Clark, 2013), where it is not enough to just learn any grammar which is equivalent to the target one. Learning PCFGs by inducing any CFG equivalent to the underlying target PCFG structure and assigning probabilities to this grammar does not guarantee that the learned PCFG is close to the target one. We need the learned structure to be somehow *close* to the target structure. Thus, the focus should be on learning the correct *grammars* rather than the correct *languages*.

– In natural language processing, PCFGs can be used as either:

  1. *Generators* of natural language distributions (i.e. describe a probability distribution over the set of all possible sentences built from a finite vocabulary).
  2. *Descriptors* of natural language structure (i.e. assign parse trees to sentences).

  Many works in the NLP literature induce PCFGs for either one of these reasons (and not both). We were more ambitious and tried to induce PCFGs which perform well as both generators of language and descriptors of its structure. In actual fact, we tried to learn PCFGs which are expected to perform well in evaluation tests for grammars induced in a supervised setting (where the learned grammars are evaluated on how well they assign parse trees to *unseen* sentences). Whilst we managed to perform well in these two tasks on artificial natural language grammars, we did not manage to come even close to the state-of-the-art in both tasks with the same induced grammar.

  Clearly, the task we embarked upon is a very difficult one, considering that learning PCFGs for unsupervised parsing or language modelling alone have themselves been hard tasks. Certainly, barely no work has been done on trying to tackle the problem as we did, and thus there is a lot of room for future work.

– Is the strategy we adopted of opting for smaller grammars enough for learning good grammars? Should other factors, apart from grammar complexity, be taken into consideration as a preference bias? If other factors need to be considered, then (as we explained in section 7.4) this can be done through the use of a weighted formula. One idea for example is to incorporate the maximum likelihood principle. This can be done by first running the inside-outside algorithm on the primitive grammar and then assigning weights to the rule variables so as to favour rules with higher probabilities assigned by the inside-outside algorithm. Another more simpler idea is to assign better weights for right-branching rules, given that natural language structure tends to have a very right-branching type of structure. On the other hand, the non-terminal variables can for example be given weights depending on the mutual information of strings contained in the non-terminal's congruence class (since, as shown in (Clark, 2001), true constitutes tend to have high mutual information between words occurring just before and after them). These and other possible ways of using a weighted formula can potentially improve the results obtained on natural language corpora.

– Our algorithm can potentially be applied in other fields, such as bioinformatics. The RNA secondary structure prediction problem in bioinformatics requires that non-complex PCFGs (in terms of size and number of parameters) are induced from only a small sample of positive data (Dowell and Eddy, 2004).

This fits perfectly into our setting, and in fact we feel that our algorithm may be better suited for this application rather than in NLP. Interestingly, in a recent work by Zhao et al. (2010), an idea similar to our reduction to MIN-SAT has been used to build tree grammars for assigning tree structures to glycan.

Another possible application of our work, which we experimented with, is in the smallest grammar problem. This is the problem of finding the smallest possible CFG describing one given string. The congruence classes for a singleton language consist of simply one class per substring, therefore we do not need the first phase of our algorithm. From these classes, we can build the formula to find a small C-CFG in CNF, which we can subsequently transform into a general non-CNF CFG by removing redundant rules. Our restriction to congruential grammars is not a drawback for the smallest grammar problem since we can easily show (as we did in Section 7.3) that any smallest grammar must be congruential. From our experiments on this problem, we obtained encouraging results on the shorter strings from the Canterbury corpus (Arnold and Bell, 1997). However, these results were not as good as the current state-of-the-art systems (Gallé, 2011). The formula was too big for the more longer strings. In spite of this, we still think there is potential for further improvement. One idea is to first deal separately (i.e. with some other method different from ours) with the problem of segmenting the whole string into small blocks, in order to build an initial production rule with a long RHS. After obtaining the initial rule, the formula can be used to select the rest of the production rules that generate each block.

– To conclude, we think that for a PCFG learning algorithm to be successful, it needs to find a way to use the statistical information in the given training sample in order to learn the underlying structure of the target PCFG. Ideally, a clear pre-defined connection is set between the observable statistical features and the unobserved structural information. The learning algorithm will then simply need to induce these features from statistical evidence and use the pre-defined link to build the grammar. In our case, we opted to link the observed congruence classes of $\cong$ (that can be induced from statistical information) to the unobserved non-terminals and production rules by trying to build a grammar which uses the smallest number of features.

Bibliography

Steven P. Abney, David A. McAllester, and Fernando Pereira. Relating Probabilistic Grammars and Automata. In Robert Dale and Kenneth Ward Church, editors, *ACL*. ACL, 1999. ISBN 1-55860-609-2. 23, 35, 58, 65

Pieter Adriaans and Ceriel Jacobs. Using mdl for grammar induction. In Yasubumi Sakakibara, Satoshi Kobayashi, Kengo Sato, Tetsuro Nishino, and Etsuji Tomita, editors, *Grammatical Inference: Algorithms and Applications*, volume 4201 of *Lecture Notes in Computer Science*, pages 293–306. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-45264-5. doi: 10.1007/11872436_24. URL http://dx.doi.org/10.1007/11872436_24. 10, 11

Pieter W. Adriaans, Marten Trautwein, and Marco Vervoort. Towards High Speed Grammar Induction on Large Text Corpora. In Václav Hlavác, Keith G. Jeffery, and Jirí Wiedermann, editors, *SOFSEM*, volume 1963 of *Lecture Notes in Computer Science*, pages 173–186. Springer, 2000. 10, 11, 12, 44, 49, 87, 97, 98

Dana Angluin. Inference of Reversible Languages. *J. ACM*, 29(3):741–765, July 1982. ISSN 0004-5411. doi: 10.1145/322326.322334. URL http://doi.acm.org/10.1145/322326.322334. 10, 11

Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987. 10, 43

Dana Angluin. Queries and Concept Learning. *Mach. Learn.*, 2(4):319–342, April 1988a. ISSN 0885-6125. doi: 10.1023/A:1022821128753. URL http://dx.doi.org/10.1023/A:1022821128753. 10

Dana Angluin. Identifying languages from stochastic examples. Technical Report YALEU/DCS/RR-614, Yale University, New Haven, CT, 1988b. 10, 35

Ross Arnold and Tim Bell. A corpus for the evaluation of lossless compression algorithms. In *Data Compression Conference, 1997. DCC'97. Proceedings*, pages 201–210. IEEE, 1997. 107

Vijay Balasubramanian. Equivalence and reduction of hidden Markov models. Master's thesis, Department of Electrical Engineering and Computer Science, MIT, 1993. issued as AI Technical Report 1370. 57

Wiktor Bartol, Joe Miró, K. Pióro, and Francesc Rosselló. On the coverings by tolerance classes. *Inf. Sci.*, 166(1-4):193–211, 2004. 84

Anja Belz. PCFG Learning by Nonterminal Partition Search. In *Proceedings of the 6th International Colloquium on Grammatical Inference: Algorithms and Applications*, ICGI '02, pages 14–27, London, UK, UK, 2002. Springer-Verlag. ISBN 3-540-44239-1. URL http://dl.acm.org/citation.cfm?id=645519.655810. 11

J-M. Benedí and J.-A. Sánchez. Estimation of stochastic context-free grammars and their use as language models. *Computer Speech & Language*, 19(3):249–274, 2005. 13

Michel Berkelaar. lpSolve: Interface to Lp solve v. 5.5 to solve linear/integer programs. *R package version*, 5(4), 2008. 90

Luc Boasson and Géraud Sénizergues. NTS Languages Are Deterministic and Congruential. *J. Comput. Syst. Sci.*, 31(3):332–342, 1985. 74, 75

Rens Bod. Unsupervised parsing with U-DOP. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, CoNLL-X '06, pages 85–92, Stroudsburg, PA, USA, 2006a. Association for Computational Linguistics. 44, 52, 103

Rens Bod. An all-subtrees approach to unsupervised parsing. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 865–872. Association for Computational Linguistics, 2006b. 45, 52, 103

Taylor L. Booth. Probabilistic representation of formal languages. In *Switching and Automata Theory, 1969., IEEE Conference Record of 10th Annual Symposium on*, pages 74–81, Oct 1969. 34

Taylor L. Booth and Richard A. Thompson. Applying Probability Measures to Abstract Languages. *IEEE Trans. Comput.*, 22(5):442–450, May 1973. ISSN 0018-9340. doi: 10.1109/T-C.1973.223746. URL http://dx.doi.org/10.1109/T-C.1973.223746. 22, 34, 35

Stefan Bordag, Christian Hänig, and Uwe Quasthoff. UnsuParse: unsupervised Parsing with unsupervised Part of Speech Tagging. In European, editor, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, 2008. 45, 52

Claus Brabrand, Robert Giegerich, and Anders Møller. *Analyzing ambiguity of context-free grammars*. Springer, 2007. 76

Rafael C. Carrasco. Accurate computation of the relative entropy between stochastic regular grammars. RAIRO *(Theoretical Informatics and Applications)*, 31(5):437–444, 1997. 58

Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *ICGI* Carrasco and Oncina (1994b), pages 139–152. ISBN 3-540-58473-0. 10

Rafael C. Carrasco and José Oncina, editors. *Grammatical Inference and Applications, Second International Colloquium, ICGI-94, Alicante, Spain, September 21-23, 1994, Proceedings*, volume 862 of *Lecture Notes in Computer Science*, 1994b. Springer. ISBN 3-540-58473-0. 110, 118

Rafael C. Carrasco and José Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *ITA*, 33(1):1–20, 1999. 68

Glenn Carroll and Eugene Charniak. *Two experiments on learning probabilistic dependency grammars from corpora*. Department of Computer Science, Univ., 1992. 52

Francisco Casacuberta and Colin de la Higuera. Computational complexity of problems on probabilistic grammars and transducers. In A. L. de Oliveira, editor, *Grammatical Inference: Algorithms and Applications, Proceedings of* ICGI *'00*, volume 1891 of LNAI, pages 15–24. Springer-Verlag, 2000. 58

Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005. 91, 92

Eugene Charniak. Statistical Parsing with a Context-Free Grammar and Word Statistics. In Benjamin Kuipers and Bonnie L. Webber, editors, *AAAI/IAAI*, pages 598–603. AAAI Press / The MIT Press, 1997. ISBN 0-262-51095-2. 35

Eugene Charniak, Mark Johnson, Micha Elsner, Joseph L. Austerweil, David Ellis, Isaac Haxton, Catherine Hill, R. Shrivaths, Jeremy Moore, Michael Pozar, and Theresa Vu. Multilevel Coarse-to-Fine PCFG Parsing. In Moore et al. (2006). 35

Qing Chen, Nicolas D. Georganas, and Emil M. Petriu. Hand gesture recognition using haar-like features and a stochastic context-free grammar. pages 1562–1571, 2008. 43

T. Chen and S. Kiefer. On the total variation distance of labelled Markov chains. In *Proceedings of LICS 2014*, 2014. doi: http://arxiv.org/abs/1405.2852. 65

Zhiyi Chi. Statistical Properties of Probabilistic Context-free Grammars. *Comput. Linguist.*, 25(1):131–160, March 1999. ISSN 0891-2017. 35

Zhiyi Chi and Stuart Geman. Estimation of Probabilistic Context-free Grammars. *Comput. Linguist.*, 24 (2):299–305, June 1998. ISSN 0891-2017. 35

Alexander Clark. *Unsupervised Language Acquisition: Theory and Practice*. PhD thesis, COGS, University of Sussex, 2001. 10, 53, 87, 90, 96, 104, 106

Alexander Clark. PAC-Learning Unambiguous NTS Languages. In Sakakibara et al. (2006), pages 59–71. ISBN 3-540-45264-8. 37, 42, 73, 79, 86, 90

Alexander Clark. Distributional Learning of Some Context-Free Languages with a Minimally Adequate Teacher. In Sempere and García (2010), pages 24–37. 10, 11, 13, 37, 42, 43, 67, 69, 73, 74, 75, 79, 90, 91, 105, 106

Alexander Clark. Towards General Algorithms for Grammatical Inference. In Marcus Hutter, Frank Stephan, Vladimir Vovk, and Thomas Zeugmann, editors, *ALT*, volume 6331 of *Lecture Notes in Computer Science*, pages 11–30. Springer, 2010b. ISBN 978-3-642-16107-0. 39

Alexander Clark. Learning trees from strings: A strong learning algorithm for some context-free grammars. *Journal of Machine Learning Research*, 14:3537–3559, 2013. URL `http://jmlr.org/papers/v14/clark13a.html`. 90, 106

Alexander Clark and Rémi Eyraud. Polynomial Identification in the Limit of Substitutable Context-free Languages. *Journal of Machine Learning Research*, 8:1725–1745, 2007. 10, 11, 37, 38, 39, 42, 43, 67, 69, 73, 75, 90

Alexander Clark and Franck Thollard. PAC-learnability of Probabilistic Deterministic Finite State Automata. *J. Mach. Learn. Res.*, 5:473–497, December 2004. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1005332.1005349`. 10

Alexander Clark, François Coste, and Laurent Miclet, editors. *Grammatical Inference: Algorithms and Applications, 9th International Colloquium, ICGI 2008, Saint-Malo, France, September 22-24, 2008, Proceedings*, volume 5278 of *Lecture Notes in Computer Science*, 2008. Springer. ISBN 978-3-540-88008-0. 119

Alexander Clark, Chris Fox, and Shalom Lappin. *The handbook of computational linguistics and natural language processing*, volume 57. John Wiley & Sons, 2010. 44, 48

Shay B. Cohen and Noah A. Smith. Empirical Risk Minimization with Approximations of Probabilistic Grammars. In John D. Lafferty, Christopher K. I. Williams, John Shawe-Taylor, Richard S. Zemel, and Aron Culotta, editors, *NIPS*, pages 424–432. Curran Associates, Inc., 2010a. 35

Shay B. Cohen and Noah A. Smith. Viterbi Training for PCFGs: Hardness Results and Competitiveness of Uniform Initialization. In *ACL*, pages 1502–1511, 2010b. 13, 35

Shay B. Cohen, Karl Stratos, Michael Collins, Dean P. Foster, and Lyle H. Ungar. Spectral Learning of Latent-Variable PCFGs. In *ACL (1)*, pages 223–231. The Association for Computer Linguistics, 2012. ISBN 978-1-937284-24-4. 13, 35

Michael Collins. *Head-driven statistical models for natural language parsing*. PhD thesis, University of Pennsylvania, 1999. 35

Craig M Cook, Azriel Rosenfeld, and Alan R Aronson. Grammatical inference by hill climbing. *Information Sciences*, 10(2):59–80, 1976. 48

C. Cortes, M. Mohri, and A. Rastogi. l_p distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, 18(4):761–779, 2007. 60

C. Cortes, M. Mohri, A. Rastogi, and M. Riley. On the computation of the relative entropy of probabilistic automata. *International Journal on Foundations of Computer Science*, 19(1):219–242, 2008. 58

François Coste, Gaëlle Garet, Jacques Nicolas, et al. Local substitutability for sequence generalization. In *ICGI 2012*, volume 21, pages 97–111. Citeseer, 2012. 67, 69

François Coste, Gaëlle Garet, and Jacques Nicolas. A bottom-up efficient algorithm learning substitutable languages from positive examples. In *The 12th International Conference on Grammatical Inference*, pages 49–63, 2014. 67, 69

François Coste and Jacques Nicolas. Inference of finite automata: Reducing the search space with an ordering of pairs of states. In Claire Nédellec and Céline Rouveirol, editors, *Machine Learning: ECML-98*, volume 1398 of *Lecture Notes in Computer Science*, pages 37–42. Springer Berlin Heidelberg, 1998. ISBN 978-3-540-64417-0. doi: 10.1007/BFb0026669. URL http://dx.doi.org/10.1007/BFb0026669. 10

Frank L. Deremer. PRACTICAL TRANSLATORS FOR LR(K) LANGUAGES. Technical report, Cambridge, MA, USA, 1969. 35

Robin D Dowell and Sean R Eddy. Evaluation of several lightweight stochastic context-free grammars for rna secondary structure prediction. *BMC bioinformatics*, 5(1):71, 2004. 106

Joost Engelfriet. Deciding the NTS property of context-free grammars. In Juliani Karhumäki, Hermann Maurer, and Grzegorz Rozenberg, editors, *Results and Trends in Theoretical Computer Science*, volume 812 of *Lecture Notes in Computer Science*, pages 124–130. Springer Berlin Heidelberg, 1994. 76

K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66, 2009a. 58

Kousha Etessami and Mihalis Yannakakis. Recursive Markov Chains, Stochastic Grammars, and Monotone Systems of Nonlinear Equations. *J. ACM*, 56(1):1:1–1:66, February 2009b. ISSN 0004-5411. doi: 10.1145/1462153.1462154. URL http://doi.acm.org/10.1145/1462153.1462154. 24

Rémi Eyraud, Colin de la Higuera, and Jean-Christophe Janodet. LARS: A learning algorithm for rewriting systems. *Machine Learning*, 66(1):7–31, 2007. 10, 11, 40

Rémi Eyraud. *Inférence Grammaticale de Langages Hors-Contextes*. PhD thesis, Université Jean Monnet de Saint-Etienne, 2006. 12

Henning Fernau and Colin de la Higuera. Grammar Induction: An Invitation to Formal Language Theorists. *Grammars*, 7:45–55, 2004. 10

V. Forejt, P. Jancar, S. Kiefer, and J. Worrell. Language equivalence of probabilistic pushdown automata. *Information and Computation*, 237:1–11, 2014. 23, 58

Matthias Gallé. *Searching for compact hierarchical structures in DNA by means of the Smallest Grammar Problem*. PhD thesis, Université Rennes 1, 2011. 107

Giorgio Gallo, Giustino Longo, and Stefano Pallottino. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42(2):177–201, 1993. doi: 10.1016/0166-218X(93)90045-P. URL http://dx.doi.org/10.1016/0166-218X(93)90045-P. 82

Pedro García, Enrique Vidal, and José Oncina. Learning Locally Testable Languages in the Strict Sense. In *ALT*, pages 325–338, 1990. 10, 11

Roland Gecse and Attila Kovács. Consistency of Stochastic Context-free Grammars. *Math. Comput. Model.*, 52(3-4):490–500, August 2010. ISSN 0895-7177. 23, 24, 35

E. Mark Gold. Language Identification in the Limit. *Information and Control*, 10(5):447–474, 1967. 10, 38

E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3): 302–320, 1978. 10

Susan L. Graham, , and Michael Harrison Walter L. Ruzzo. An improved context-free recognizer. *ACM Trans. Program. Lang. Syst.*, 2(3):415–462, July 1980. ISSN 0164-0925. doi: 10.1145/357103.357112. URL http://doi.acm.org/10.1145/357103.357112. 29

Leslie Grate. Automatic RNA Secondary Structure Determination with Stochastic Context-Free Grammars. In Christopher J. Rawlings, Dominic A. Clark, Russ B. Altman, Lawrence Hunter, Thomas Lengauer, and Shoshana J. Wodak, editors, *ISMB*, pages 136–144. AAAI, 1995. ISBN 0-929280-83-0. 35

Ulf Grenander. *Syntax-controlled Probabilities*. Division of Applied Mathematics, Brown University, 1967. 34

Peter D. Grünwald. *The minimum description length principle*. The MIT Press, 2007. 46

M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978. ISBN 0201029553. 21

Marijn J.H. Heule and Sicco Verwer. Exact dfa identification using sat solvers. In José M. Sempere and Pedro García, editors, *Grammatical Inference: Theoretical Results and Applications*, volume 6339 of *Lecture Notes in Computer Science*, pages 66–79. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-15488-1_7. 10

Colin de la Higuera. Characteristic Sets for Polynomial Grammatical Inference. *Machine Learning*, 27(2): 125–138, 1997. 10, 38, 39

Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. 2010. 10, 12, 13, 40, 41, 43, 57, 83

Colin de la Higuera and J. Oncina. The most probable string: an algorithmic study. *Journal of Logic and Computation*, doi: 10.1093/logcom/exs049, 2013a. 58

Colin de la Higuera and J. Oncina. Computing the most probable string with a probabilistic finite state machine. In *Proceedings of* FSMNLP, 2013b. 58, 65

Colin de la Higuera and José Oncina. Inferring Deterministic Linear Languages. In Jyrki Kivinen and Robert H. Sloan, editors, *COLT*, volume 2375 of *Lecture Notes in Computer Science*, pages 185–200. Springer, 2002. ISBN 3-540-43836-X. 10, 11, 37, 40

Colin de la Higuera and José Oncina. Identification with Probability One of Stochastic Deterministic Linear Languages. In Ricard Gavaldà, Klaus P. Jantke, and Eiji Takimoto, editors, *ALT*, volume 2842 of *Lecture Notes in Computer Science*, pages 247–258. Springer, 2003. ISBN 3-540-20291-9. 10, 37, 41

Colin de la Higuera and José Oncina. Learning Stochastic Finite Automata. In Paliouras and Sakakibara (2004), pages 175–186. 40, 41

Jan Anne Hogendorp. Controlled bidirectional grammars. *International Journal of Computer Mathematics*, 27(3-4):159–180, 1989. 76

James Jay Horning. *A Study of Grammatical Inference*. PhD thesis, 1969. 34

T. Huang and Kingsun Fu. On Stochastic Context-free Languages. *Inf. Sci.*, 3(3):201–224, July 1971. ISSN 0020-0255. 34

Hiroki Ishizaka. Polynomial Time Learnability of Simple Deterministic Languages. *Mach. Learn.*, 5(2): 151–164, July 1990. ISSN 0885-6125. doi: 10.1023/A:1022644732619. URL http://dx.doi.org/10.1023/A:1022644732619. 43

A. Jagota, R. B. Lyngsø, and C. N. S. Pedersen. Comparing a hidden Markov model and a stochastic context-free grammar. In *Proceedings of* WABI *'01*, number 2149 in LNCS, pages 69–84. Springer-Verlag, 2001. 58, 60, 62

Frederick Jelinek and John D. Lafferty. Computation of the Probability of Initial Substring Generation by Stochastic Context-Free Grammars. *Computational Linguistics*, 17(3):315–323, 1991. 30, 35

Frederick Jelinek, John D Lafferty, and Robert L Mercer. *Basic methods of probabilistic context free grammars*. Springer, 1992. 35

Mark Johnson. PCFG Models of Linguistic Tree Representations. *Comput. Linguist.*, 24(4):613–632, December 1998. ISSN 0891-2017. 13, 35

Mark Johnson, Thomas L. Griffiths, and Sharon Goldwater. Bayesian Inference for PCFGs via Markov Chain Monte Carlo. In *HLT-NAACL*, pages 139–146, 2007. 13, 35

Daniel Jurafsky, Chuck Wooters, Jonathan Segal, Andreas Stolcke, Eric Fosler, G Tajchaman, and Nelson Morgan. Using a stochastic context-free grammar as a language model for speech recognition. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 189–192. IEEE, 1995. 13, 35

Anna Kasprzik and Ryo Yoshinaka. Distributional learning of simple context-free tree grammars. In Jyrki Kivinen, Csaba Szepesvári, Esko Ukkonen, and Thomas Zeugmann, editors, *ALT*, volume 6925 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2011. ISBN 978-3-642-24411-7. 96

Martin Kay. Chart Generation. In *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ACL '96, pages 200–204, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics. doi: 10.3115/981863.981890. URL http://dx.doi.org/10.3115/981863.981890. 35

Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *J. ACM*, 41(1):67–95, January 1994. ISSN 0004-5411. doi: 10.1145/174644.174647. URL http://doi.acm.org/10.1145/174644.174647. 10

Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, Cambridge, MA, USA, 1994. ISBN 0-262-11193-4. 46

Dan Klein. *The Unsupervised Learning of Natural Language Structure*. PhD thesis, Stanford University, 2004. 13, 44, 45, 52, 90, 103

Dan Klein and Christopher D. Manning. A generative constituent-context model for improved grammar induction. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 128–135, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073106. URL http://dx.doi.org/10.3115/1073083.1073106. 11, 52

Donald E. Knuth. On the Translation of Languages from Left to Rigth. *Information and Control*, 8(6): 607–639, 1965. 35

W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Springer-Verlag, 1986. 21

Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. In Vasant Honavar and Giora Slutzki, editors, *ICGI*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998. ISBN 3-540-64776-7. 10, 11, 86

Pat Langley and Sean Stromsten. Learning Context-Free Grammars with a Simplicity Bias. In Ramon López de Mántaras and Enric Plaza, editors, *ECML*, volume 1810 of *Lecture Notes in Computer Science*, pages 220–228. Springer, 2000. 11, 48, 97, 98

Karim Lari and Steve J. Young. The estimation of stochastic context-free grammars using the Inside-Outside algorithm. *Computer Speech & Language*, 4(1):35 – 56, 1990. 13, 34, 35, 52, 91

J. A. Laxminarayana and G. Nagaraja. Inference of a Subclass of Context Free Grammars Using Positive Samples. In Colin de la Higuera, Pieter W. Adriaans, Menno van Zaanen, and José Oncina, editors, *ECML Workshop on Learning Contex-Free Grammars*, pages 29–40. Ruder Boskovic Institute, Zagreb, Croatia, 2003. ISBN 953-6690-39-X. 10, 39

Franco M. Luque and Gabriel G. Infante López. Bounding the Maximal Parsing Performance of Non-Terminally Separated Grammars. In Sempere and García (2010), pages 135–147. 103

R. B. Lyngsø and C. N. S. Pedersen. Complexity of comparing hidden Markov models. In *Proceedings of* ISAAC *'01*, number 2223 in LNCS, pages 416–428. Springer-Verlag, 2001. 58

R. B. Lyngsø and C. N. S. Pedersen. The consensus string problem and the complexity of comparing hidden markov models. *Journal of Computing and System Science*, 65(3):545–569, 2002. 58

R. B. Lyngsø, C. N. S. Pedersen, and H. Nielsen. Metrics and similarity measures for hidden Markov models. In *Proceedings of* ISMB *'99*, pages 178–186, 1999. 58, 62

Madhav V. Marathe and S. S. Ravi. On approximation algorithms for the minimum satisfiability problem. pages 23–29, 1996. 94

Takuya Matsuzaki, Yusuke Miyao, and Jun'ichi Tsujii. Probabilistic CFG with Latent Annotations. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 75–82, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. doi: 10.3115/1219840.1219850. URL http://dx.doi.org/10.3115/1219840.1219850. 35

Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997. ISBN 0070428077, 9780070428072. 9

Darnell J. Moore and Irfan A. Essa. Recognizing multitasked activities from video using stochastic context-free grammar. In Rina Dechter and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.*, pages 770–776. AAAI Press / The MIT Press, 2002. URL http://www.aaai.org/Library/AAAI/2002/aaai02-116.php. 43

Robert C. Moore, Jeff A. Bilmes, Jennifer Chu-Carroll, and Mark Sanderson, editors. *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 4-9, 2006, New York, New York, USA*, 2006. The Association for Computational Linguistics. 111, 116

T. Murgue and Colin de la Higuera. Distances between distributions: Comparing language models. In A. Fred, T. Caelli, R. Duin, A. Campilho, and D. de Ridder, editors, *Structural, Syntactic and Statistical Pattern Recognition, Proceedings of* SSPR *and* SPR *2004*, volume 3138 of LNCS, pages 269–277. Springer-Verlag, 2004. 58

Katsuhiko Nakamura and Takashi Ishiwata. Synthesizing Context Free Grammars from Sample Strings Based on Inductive CYK Algorithm. In Arlindo L. Oliveira, editor, *Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Computer Science*, pages 186–195. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-41011-9. doi: 10.1007/978-3-540-45257-7_15. URL http://dx.doi.org/10.1007/978-3-540-45257-7_15. 10

M-J. Nederhof and G. Satta. Kullback-leibler distance between probabilistic context-free grammars and probabilistic finite automata. In *Proceedings of* COLING *'04 Proceedings of the 20th international conference on Computational Linguistics*, number 71, 2004. 58

Mark-Jan Nederhof and Giorgio Satta. Estimation of Consistent Probabilistic Context-free Grammars, booktitle = HLT-NAACL. In Moore et al. (2006). 35

Mark-Jan Nederhof and Giorgio Satta. Computation of distances for regular and context-free probabilistic languages. *Theor. Comput. Sci.*, 395(2-3):235–254, 2008. 35

Mark-Jan Nederhof and Giorgio Satta. Computing Partition Functions of PCFGs. *Res. Lang. Comput.*, 7(2-4):233–233, December 2009. ISSN 1570-7075. doi: 10.1007/s11168-009-9062-1. URL http://dx.doi.org/10.1007/s11168-009-9062-1. 35

Mark-Jan Nederhof and Giorgio Satta. Computation of Infix Probabilities for Probabilistic Context-Free Grammars. In *EMNLP*, pages 1213–1221. ACL, 2011. ISBN 978-1-937284-11-4. 35

Anil Nerode. Linear Automaton Transformation. In *Proc. American Mathematical Society*, volume 9, pages 541–544, 1958. 68

José Oncina and Pedro García. Inferring Regular Languages in Polynomial Update Time. *Pattern Recognition and Image Analysis*, pages 49–61, 1992. 10, 11

Georgios Paliouras and Yasubumi Sakakibara, editors. *Grammatical Inference: Algorithms and Applications, 7th International Colloquium, ICGI 2004, Athens, Greece, October 11-13, 2004, Proceedings*, volume 3264 of *Lecture Notes in Computer Science*, 2004. Springer. 114, 117

Fernando C. N. Pereira and David H. D. Warren. Parsing As Deduction. In *Proceedings of the 21st Annual Meeting on Association for Computational Linguistics*, ACL '83, pages 137–144, Stroudsburg, PA, USA, 1983. Association for Computational Linguistics. doi: 10.3115/981311.981338. URL http://dx.doi.org/10.3115/981311.981338. 35

Georgios Petasis, Georgios Paliouras, Constantine D. Spyropoulos, and Constantine Halatsis. eg-GRIDS: Context-Free Grammatical Inference from Positive Examples Using Genetic Search. In Paliouras and Sakakibara (2004), pages 223–234. 10, 11, 44, 47, 48

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning Accurate, Compact, and Interpretable Tree Annotation. In Nicoletta Calzolari, Claire Cardie, and Pierre Isabelle, editors, *ACL*. The Association for Computer Linguistics, 2006. 35

E. L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946. 60

Elaine Rich. *Automata, computability and complexity: theory and applications*. Pearson Prentice Hall Upper Saddle River, 2008. 21

James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. On languages piecewise testable in the strict sense. In *Proceedings of the 10th and 11th Biennial conference on The mathematics of language*, MOL'07/09, pages 255–265, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-14321-0, 978-3-642-14321-2. URL http://dl.acm.org/citation.cfm?id=1886644.1886663. 11

Dana Ron, Yoram Singer, and Naftali Tishby. On the learnability and usage of acyclic probabilistic finite automata. In *Proceedings of the eighth annual conference on Computational learning theory*, COLT '95, pages 31–40, New York, NY, USA, 1995. ACM. ISBN 0-89791-723-5. doi: 10.1145/225298.225302. URL http://doi.acm.org/10.1145/225298.225302. 10

Wojciech Rytter. Application of lempel-ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.*, 302(1-3):211–222, 2003. 91, 92

Y. Sakakibara, M. Brown, R. Hughley, I. Mian, K. Sjolander, R. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nuclear Acids Research*, 22:5112–5120, 1994a. 43

Yasubumi Sakakibara. On learning from queries and counterexamples in the presence of noise. *Information Processing Letters*, 37(5):279–284, 1991. 10

Yasubumi Sakakibara. Learning context-free grammars using tabular representations. *Pattern Recogn.*, 38 (9):1372–1383, September 2005. ISSN 0031-3203. doi: 10.1016/j.patcog.2004.03.021. URL http://dx.doi.org/10.1016/j.patcog.2004.03.021. 11

Yasubumi Sakakibara, Michael Brown, Richard Hughey, I. Saira Mian, Kimmen Sjölander, Rebecca C. Underwood, and David Haussler. Recent Methods for RNA Modeling Using Stochastic Context-Free Grammars. In Maxime Crochemore and Dan Gusfield, editors, *CPM*, volume 807 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 1994b. ISBN 3-540-58094-8. 10, 13, 35

Yasubumi Sakakibara, Satoshi Kobayashi, Kengo Sato, Tetsuro Nishino, and Etsuji Tomita, editors. *Grammatical Inference: Algorithms and Applications, 8th International Colloquium, ICGI 2006, Tokyo, Japan, September 20-22, 2006, Proceedings*, volume 4201 of *Lecture Notes in Computer Science*, 2006. Springer. ISBN 3-540-45264-8. 111, 119

Arto Salomaa. Probabilistic and weighted grammars. *Information and Control*, 15(6):529 – 544, 1969. 34

Ismael Salvador and Jose-Miguel Benedi. Rna modeling by combining stochastic context-free grammars and n-gram models. *International Journal of Pattern Recognition and Artificial Intelligence*, 16(03): 309–315, 2002. 13

Joan-Andreu Sánchez and José-Migual Benedí. Consistency of Stochastic Context-Free Grammars From Probabilistic Estimation Based on Growth Transformations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19 (9):1052–1055, September 1997. ISSN 0162-8828. 35

Yoav Seginer. Fast Unsupervised Incremental Parsing. In John A. Carroll, Antal van den Bosch, and Annie Zaenen, editors, *ACL*. The Association for Computational Linguistics, 2007. 44, 45, 52, 103

José M. Sempere and Pedro García. A Characterization of Even Linear Languages and its Application to the Learning Problem. In Carrasco and Oncina (1994b), pages 38–44. ISBN 3-540-58473-0. 10, 11, 40

José M. Sempere and Pedro García, editors. *Grammatical Inference: Theoretical Results and Applications, 10th International Colloquium, ICGI 2010, Valencia, Spain, September 13-16, 2010. Proceedings*, volume 6339 of *Lecture Notes in Computer Science*, 2010. Springer. 111, 115, 119

Geraud Senizergues. The equivalence and inclusion problems for NTS languages. *Journal of Computer and System Sciences*, 31(3):303–331, 1985. 76

Chihiro Shibata and Ryo Yoshinaka. Pac learning of some subclasses of context-free grammars with basic distributional properties from positive data. In *ALT*, pages 143–157, 2013. 79, 86, 90

Hiromi Shirakawa and Takashi Yokomori. Polynomial-time MAT Learning of C-Deterministic Context-free Grammars", journal = "Journal of Information Processing. 15(EX):42–52, 1993. 43

Zach Solan, David Horn, Eytan Ruppin, and Shimon Edelman. Unsupervised Learning of Natural Languages. *Proceedings of the National Academy of Sciences of the United States of America*, 102(33): 11629–11634, 2005. 11, 12, 44, 53, 97, 98, 100, 123

Gilbert W. Stewart. *Introduction to Matrix Computations*. Academic Press, New York, 1973. 24

Andreas Stolcke. *Bayesian learning of probabilistic language models*. PhD thesis, University of California, Berkeley, 1994. 10, 13, 35, 44, 47, 48, 90, 97, 98

Andreas Stolcke. An Efficient Probabilistic Context-Free Parsing Algorithm that Computes Prefix Probabilities. *Computational Linguistics*, 21(2):165–201, 1995. 28, 29, 35

Patrick Suppes. Probabilistic grammars for natural languages. *Synthese*, 22(1):95–116, 1970. 34

Isabelle Tellier. Grammatical inference by specialization as a state splitting strategy. In *proceedings of the 16th Amsterdam colloquium*, pages 223–228, Netherlands, 2007. URL http://hal.archives-ouvertes.fr/hal-00470267. 11

Masaru Tomita. *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 1985. ISBN 0898382025. 35

Menno van Zaanen. *Bootstrapping Structure into Language: Alignment-Based Learning*. PhD thesis, University of Leeds, 2001. 10, 11, 13, 44, 45, 49, 50, 51, 87, 97, 99

Menno van Zaanen and Pieter Adriaans. Alignment-based learning versus emile: a comparison. In *Proceedings of the Belgian-Dutch Conference on Artificial Intelligence (BNAIC); Amsterdam, the Netherlands*. Citeseer, 2001. 52, 121

Menno van Zaanen and Jeroen Geertzen. Problems with Evaluation of Unsupervised Empirical Grammatical Inference Systems. In Clark et al. (2008), pages 301–303. ISBN 978-3-540-88008-0. 45, 99

Menno van Zaanen and Nanne van Noord. Model merging versus model splitting context-free grammar induction. In Jeffrey Heinz, Colin de la Higuera, and Tim Oates, editors, *Proceedings of the Eleventh International Conference on Grammatical Inference, ICGI 2012, University of Maryland, College Park, USA, September 5-8, 2012*, volume 21 of *JMLR Proceedings*, pages 224–236. JMLR.org, 2012. URL http://jmlr.csail.mit.edu/proceedings/papers/v21/vanzaanen12b.html. 53

Enrique Vidal, Frank Thollard, Colin De La Higuera, Francisco Casacuberta, and Rafael C Carrasco. Probabilistic finite-state machines-part ii. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(7):1026–1039, 2005. 79

Heidi R. Waterfall, Ben Sandbank, Luca Onnis, and Shimon Edelman. An empirical generative framework for computational modeling of language acquisition. *Journal of Child Language*, 37:671–703, 6 2010. 53, 97, 100

Charles S. Wetherell. Probabilistic Languages: A Review and Some Open Questions. *ACM Comput. Surv.*, 12(4):361–379, 1980. 22, 23, 24

J. Gerard Wolff. Grammar Discovery as Data Compression. In *AISB/GI (ECAI)*, pages 375–379, 1978. 48

Takashi Yokomori. Polynomial-time identification of very simple grammars from positive data. *Theor. Comput. Sci.*, 1(298):179–206, 2003. 10, 11, 39

Ryo Yoshinaka. Polynomial-Time Identification of an Extension of Very Simple Grammars from Positive Data. In Sakakibara et al. (2006), pages 45–58. ISBN 3-540-45264-8. 39

Ryo Yoshinaka. Identification in the Limit of k, l-Substitutable Context-Free Languages. In Clark et al. (2008), pages 266–279. ISBN 978-3-540-88008-0. 39, 67, 69, 90

Ryo Yoshinaka. Polynomial-time identification of multiple context-free languages from positive data and membership queries. In Sempere and García (2010), pages 230–244. 11, 67, 69

Ryo Yoshinaka and Alexander Clark. Polynomial time learning of some multiple context-free languages with a minimally adequate teacher. In Philippe de Groote and Mark-Jan Nederhof, editors, *FG*, volume 7395 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2010. ISBN 978-3-642-32023-1. 96

Yang Zhao, Morihiro Hayashida, and Tatsuya Akutsu. Integer programming-based method for grammar-based tree compression and its application to pattern extraction of glycan tree structures. *BMC bioinformatics*, 11(Suppl 11):S4, 2010. 107

List of Tables

List of Figures

List of Algorithms

l'université nantes angers le mans
COMMUNAUTÉ D'UNIVERSITÉS ET ÉTABLISSEMENTS

UNIVERSITÉ DE NANTES

Thèse de Doctorat

James SCICLUNA

Inférence grammaticale de grammaires incontextuelles probabilistes

Grammatical Inference of Probabilistic Context-Free Grammars

Résumé

L'inférence grammaticale consiste à apprendre, à partir de données provenant d'un langage, une grammaire susceptible d'expliquer ou de générer le langage en question. Ce travail, concerne les grammaires incontextuelles (ou context-free) probabilistes, plus puissantes que les grammaires régulières, objet de la plupart des travaux en inférence grammaticale. L'apprentissage est non supervisé : aucune information structurelle n'est connue. Le travail comprend un état de l'art concernant l'inférence grammaticale, les grammaires probabilistes et les classes de grammaires permettant un apprentissage distributionnel. Puis nous étudions différents problèmes de décision concernant des questions de (calculs de) distances entre distributions et nous montrons qu'en général il s'agit de problèmes indécidables. Dans un second temps nous donnons une description mathématique de la classe de grammaires qui vont nous intéresser. Le cœur de la thèse concerne le développement de l'algorithme COMINO, de l'analyse de ses propriétés et de l'étude empirique de ses capacités. L'algorithme se déroule en trois phases : durant la première, une relation d'équivalence sur les sous-mots est calculée. Durant la seconde, un solveur est utilisé pour sélectionner un nombre minimal de classes. Enfin, les classes deviennent les nonterminaux d'une grammaire dont les poids des règles sont estimés grâce à l'échantillon. Les résultats expérimentaux témoignent de la robustesse de l'approche mais montrent également les limites de l'approche sur des données réelles de langue naturelle.

Mots clés

Inférence grammaticale , grammaires probabilistes , apprentissage automatique

Abstract

Probabilistic Context-Free Grammars (PCFGs) are formal statistical models which describe probability distributions on strings and on tree structures of the same strings. Grammatical Inference is a sub-field of machine learning where the task is to learn automata or grammars (such as PCFGs) from information about their languages. In this thesis, we are interested in Grammatical Inference of PCFGs from text. There are various applications for this problem, chief amongst which are Unsupervised Parsing and Language Modelling in Natural Language Processing and RNA secondary structure prediction in Bioinformatics. PCFG inference is however a difficult problem for a variety of reasons. In spite of its importance for various applications, only few positive results have up till now been obtained for this problem.

Our main contribution in this thesis is a practical PCFG learning algorithm with some proven properties and based on a principled approach. We define a new subclass of PCFGs (very similar to the one defined in (Clark, 2010)) and use distributional learning and MDL-based techniques in order to learn this class of grammars. We obtain competitive results on experiments that evaluate unsupervised parsing and language modelling.

A minor contribution in this thesis is a compendium of undecidability results for distances between PCFGs along with two positive results on PCFGs. Having such results can help in the process of finding learning algorithms for PCFGs.

Key Words

Grammatical inference, Probabilistic context-free grammars, Machine learning