Application of a Stochastic Grammatical Inference Method to Text Structure

Matthew Young-Lai

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Technical Report CS-96-36[1]

October 1, 1996

[1]This report is a reprint of a thesis that embodies the results of research done in partial fulfillment of the requirements for the degree of Master of Mathematics in Computer Science at the University of Waterloo.

Abstract

For a document collection where structural elements are identified with markup, it is sometimes necessary to retrospectively construct a grammar that constrains element nesting and ordering. An approach to this problem is described based on a grammatical inference method that generates stochastic finite automata. Improvements are made to the algorithm based on experiments with data from the *Oxford English Dictionary*. The problems of understanding results and interactively adjusting the algorithm's parameters are also considered.

Acknowledgements

I would like to thank my supervisor Dr. Frank Tompa and the two readers of my thesis Dr. Ming Li and Dr. Derick Wood.

Financial assistance from the Natural Sciences and Engineering Research Council of Canada through a postgraduate scholarship, the Information Technology Research Center of Ontario, and the University of Waterloo is gratefully acknowledged.

Contents

List of Figures

Chapter 1

Introduction

Text, whether stored electronically, printed in a book or spray-painted on the wall of an alley, can always be thought of as having some level of structure, and this structure can be thought of as following certain rules. The term *structure* as applied to text refers to the organization expressed by grouping words into phrases, sentences, paragraphs, sections, chapters, titles, headings, footnotes, or generally, any of the various elements used to subdivide documents. The rules that these elements follow take the form of restrictions on how and where the elements can be used. A title element, for example, should not normally contain other sub-elements such as paragraphs, while a chapter element may admit several levels of sub-nesting.

Structural elements can be identified implicitly in text through spacing and punctuation patterns, font changes or typesetting instructions, while rules constraining the nesting and ordering of elements can be implied by how the elements are used. Many manipulations of computerized documents stored for purposes such as information retrieval or electronic publishing, however, first require that such implicit information be interpreted and understood. In such situations, there are well known advantages to having the information specified explicitly [AFQ89, Ber89, CRD87]. Several standards exist for including explicit structural information in documents. Of these, SGML (standard generalized markup language) [ISO86] is the most widely known and used.

The standard way to identify structural elements explicitly in a document is to interleave the text with appropriately labeled tags. The following text, for example, contains SGML-style tags that identify the entire fragment as a quotation and the sub-elements as a reference and three sentences.

```
<quotation>
<reference>The Art of War: Chapter 3 paragraph 18<\reference>
<sentence>
If you know the enemy and know yourself, you need not fear the
result of a hundred battles. <\sentence>
<sentence>
If you know yourself but not the enemy, for every victory gained
you will also suffer a defeat. <\sentence>
<sentence>
If you know neither the enemy nor yourself, you will succumb in
every battle. <\sentence>
<\quotation>
```

Documents marked up in this way can be updated and interpreted much more robustly than if the structural elements are identified with codes specific to a particular system or typesetting style. It can also be used to support operations such as searches that require words to occur within particular elements.

The standard way to express the rules that govern ordering or nesting of structural elements is with a grammar. The above example, for instance, can be thought of as conforming to the following grammar represented as a regular expression:

$$\text{quotation} \rightarrow \text{reference sentence}^{+}.$$

In SGML, a grammar is specified in the form of a *document type definition* file or DTD. Grammars can be used to verify that newly created documents conform to an existing model; to perform searches for specific structural combinations; or, generally, to perform operations that require an understanding of a document's structure. A grammar for a collection of documents can be thought of as serving much the same purpose as a schema for a traditional database: it provides an overall description of how the data is organized.

Many documents are created with their structural elements and the grammar controlling these elements only implicitly specified. This gives rise to the problem of making both these types of information explicit. The recognition of structural elements involves converting forms of markup such as typesetting instructions to descriptive tags. This requires an understanding of the original conventions used to map element types to their current layout. Approaches have been based on interactive systems [FX93, KLMN90], and

manual trial and error construction of finite state transducers [Kaz86, Cla91]. A second problem involves extracting implicit structural rules from documents and representing them as a grammar. This requires that the original intentions of the author be reconstructed by extrapolating from the available examples in some appropriate way.

This thesis presents a novel approach to the problem of automatically generating a grammar for the structure of a collection of documents. This can be considered an application of *grammatical inference* — the general problem that deals with constructing grammars consistent with training data. Grammatical inference is a difficult problem because of several factors:

- It is usual for the target language to be infinite, while the training data is always finite, i.e. many strings that should be allowed by the model will not be present in the training data.

- Infinitely many possible grammars will be consistent with a given finite sample. Some way is needed not just to construct a possible grammar, but to choose from among the possibilities.

- Some of the restrictions regarding what constitutes a good result may be difficult to evaluate objectively or automatically. For example, a grammar may have to be understandable to a human reader.

Unlike previous approaches to grammatical inference for text structure, the method described here uses frequency information associated with the examples to produce a *stochastic grammar* — one that associates a probability distribution with the strings of the generated language. The specific method that we have chosen to use is based on a grammatical inference algorithm presented in ICGI 94 [CO94b] and modified here with some generally applicable improvements.

Stochastic grammatical inference is usually applied to problems where the probabilistic accuracy of the resulting model is the primary consideration. For text structure, however, the understandability of the model is at least as important. For this reason, some additional mention is made here of techniques for understanding and visualizing stochastic grammars, as well as using this understanding to tune the parameters of the inference method.

Experiments and testing were done using data from the computerized version of the *Oxford English Dictionary (OED)* [MBCO33]. This is an extremely large document with complex structure [Ber93], containing over sixty

types of elements which are heavily nested in over two million element instances. It has been converted from its original printed form, through an intermediate keyed form with typesetting information, to a computerized form with explicitly tagged structural elements [Kaz86]. No grammar is explicitly defined. The text is broken up into roughly two hundred and ninety thousand top level elements (dictionary entries). The grammar inference problem can be considered as one of finding a complete grammar to describe these top level entries, or it can be broken into many subproblems of finding grammars for lower level elements such as quotation paragraphs or etymologies.

The rest of this thesis is organized as follows: Chapters 2 and 3 are background. They overview grammatical inference in general and its application specifically to text structure. Chapter 4 is a complete description of the inference method investigated in this work. Chapter 5 describes techniques for understanding results in the form of stochastic grammars and using this understanding to tune parameters of the inference method. Chapter 6 presents experimental results. Chapter 7 gives conclusions and suggestions for future work.

Chapter 2

Grammatical Inference

2.1 Introduction

Grammatical inference encompasses theory and methods concerned with the problem of learning grammars from training data [Vid94]. It is a well-established discipline that dates back to the sixties [Gol67, Hor69], thus predating the broader field referred to today as *machine learning* [Ang92] but still clearly falling within its boundaries. Specifically, grammatical inference is classified by machine learning as a type of *inductive inference*, the term given to learning techniques that try to guess general rules from examples.

The traditional application domain of grammatical inference has been syntactic pattern recognition. Here it is assumed that pattern classes can be represented by grammatical models, and it is often useful to be able to learn these models automatically. Other application areas have included speech and natural language processing, information retrieval, gene analysis, sequence prediction, cryptography and compression.

The basic problem is to find a grammar consistent with a given training set of positive examples. The situation where both positive and negative examples are available is also sometimes considered, although this does not correspond to most real applications. In either case, inference must usually be concerned with more than just consistency with the training data. To be useful, a grammar must generalize outside of the available examples in some reasonable way.

This chapter is a brief overview of grammatical inference. Section 2.2

outlines theoretical learnability results that have been developed; Section 2.3 lists grammar models that have been used as target representations; Section 2.4 discusses constructing default models in these representations to be used as starting points for inference; Sections 2.5 and 2.6 discuss probabilistic and non-probabilistic inference approaches.

2.2 Theoretical Learnability Frameworks

The classic formalization of grammatical inference, introduced by Gold in 1967 [Gol67], is known as *language identification in the limit*. Two sets of strings are assumed: $R+$ the positive examples, and $R-$ the negative examples. A language is said to be *identifiable in the limit* if it can be learned by *some method* for *sufficiently large* $R+$ and $R-$. Put another way, adding new examples to $R+$ and $R-$ must only produce a finite number of changes to the hypothesized model.

Two decidability results exist within this framework. The first is negative and says that no infinite language can be identified in the limit from only positive examples. This is intuitively a consequence of over-generalization since adding more positive examples can never imply that a string mistakenly included in the model should subsequently be removed. In fact, a consistent generalization of any set of positive examples would be a language containing all finite strings constructible using the alphabet. The second decidability result is positive and states that any member of an enumerable class of recursive languages (context-sensitive and below) is identifiable in the limit given both positive and negative data.

While of interest from a theoretical point of view, this framework has limited applicability to practical problems. This is primarily because no concrete, finite bound is given defining what a *sufficiently large* sample is. The first decidability result would therefore not have been very constructive even if reversed. The second is also not useful since negative examples are usually not available in real applications, and even if they are, their use can lead to intractable problems. Finding the smallest finite automaton compatible with given positive and negative example sets, for instance, has been shown to be NP-hard [Ang78, Gol78]. A polynomial time algorithm has recently been introduced that constructs a *non-minimal* solution to this problem in the limit [Oa92], however, the difficulties remain that negative examples are required and that no guarantees are made about just how big

the sample sizes might need to be.

Alternative frameworks have been developed that explicitly assume the stochastic nature of the grammars being inferred and are applicable to learning schemes based on probabilistic criteria. Using probabilistic information to discover rules is in fact a basic paradigm in the field of *information theory*, and Shannon established very early that language learning can be seen as a problem of estimating probabilities of sequences of n symbols or n-grams as n approaches infinity. Another framework for approximate learning is *probably approximately correct* (PAC) identification [Val84]. Here, a learning method is evaluated in terms of how well it can find models which, with arbitrarily high probability, are arbitrarily precise approximations to an unknown data source. Overall, it can be argued that probabilistic frameworks are more applicable to the practical requirements of real situations than other criteria such as identification in the limit.

2.3 Target Representations

Before the problem of generating a grammar can be considered, an appropriate grammar representation must be chosen. The main consideration in this choice is the standard tradeoff between the expressive power of the representation and the difficulty of the inference process. Context free grammars, for example, are more powerful than regular grammars, but their inference is much more difficult and computationally expensive.

Within the standard Chomsky hierarchy studied in automata theory [HU79], grammatical inference research has focused mainly on regular grammars because of their relative simplicity. Restricted regular grammars, usually referred to as *characterizable* subclasses, are also commonly targeted. k-reversible languages [Ang82], for example, generalize from 0-reversible languages which are those generated by finite automata that remain deterministic if all of their arcs are reversed. The main justification for using restricted language classes is to reduce the time complexity of the inference process. The defining restrictions do not themselves usually have any relevance to real world applications.

Relatively little work has been done on inference of higher level grammars. Some research has been done on inferring non-regular subclasses of context free grammars, and on inferring context free grammars given training data consisting of positive examples plus additional information in the form of

unlabeled derivation trees.

The final requirement of many practical applications is a grammar that assigns a probability distribution to the strings of the generated language. Such a *stochastic* grammar can be viewed as being composed of separate structural and probabilistic components. The structural component is just a standard grammar. The probabilistic component specifies the probabilities associated with individual strings. This can be done by assigning a probability to every production of the grammar, or equivalently for regular grammars, by assigning a probability to every transition of the associated finite automaton. The probability of a string is then the product of the production or transition probabilities used in its generation. Note that non-deterministic stochastic finite automata are sometimes referred to as *Markov models* or *hidden Markov models* in the context of probabilistic modeling.

2.4 Inference Paradigms

Most grammatical inference algorithms use an approach of starting with a model chosen so as to accept the finite language composed of exactly those strings that are in the training set. This *de-facto* model is highly specific and the inference process operates by applying generalization operations to transform it into a final result. Such a paradigm can be seen as analogous to a natural learning process where the "model" starts off as a complete memory of specific examples and large enough groups of similar examples are then combined into general rules.

The form of the de-facto model depends on the target representation. The training set $\{abb, bb, aabb, abba\}$, for example, can be represented by the de-facto grammar in Figure 2.1, the de-facto non-deterministic finite automaton in Figure 2.2, the de-facto deterministic finite automaton in the Figure 2.3, the de-facto deterministic stochastic finite automaton in Figure 2.4, or by appropriately chosen models in other representations.

Generalization operations conceptually operate by combining specific examples into descriptions that cover those examples and possibly others with similar characteristics that are not present in the training data. Of course, operations that arbitrarily expand the language generated by the model (by adding symbols to the alphabet or adding strings that have nothing in common with any of the training examples) are also possible, but these are not useful for inference purposes. Therefore, all generalization operations of in-

$$
\begin{aligned}
S &\rightarrow abb \\
S &\rightarrow bb \\
S &\rightarrow aabb \\
S &\rightarrow abba
\end{aligned}
$$

Figure 2.1: A de-facto grammar.

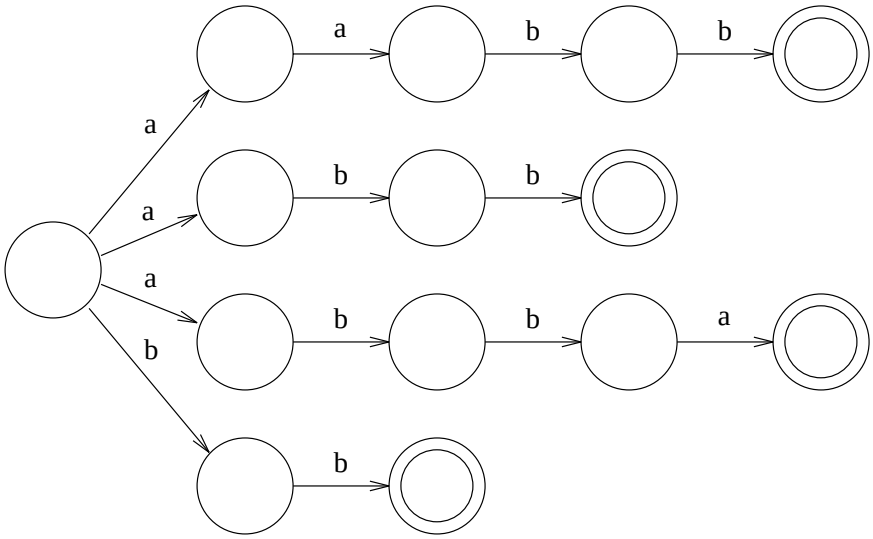

Figure 2.2: A de-facto non-deterministic finite automaton.

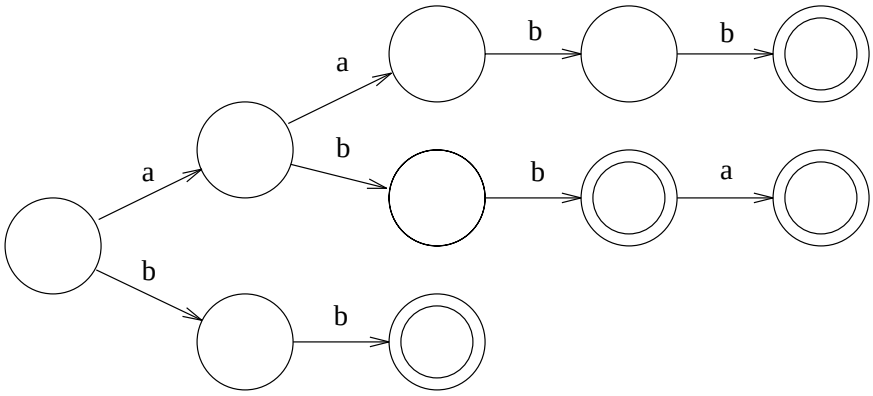

Figure 2.3: A de-facto deterministic finite automaton.

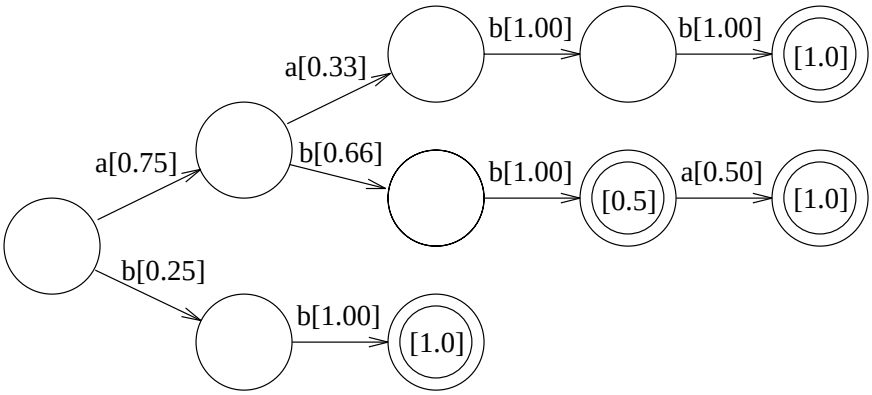

Figure 2.4: A de-facto stochastic finite automaton.

terest essentially just merge sub-components of the model, even if they are not explicitly formulated as merges. Finite automata, for example, are always generalized by merging states, thus creating new loops or paths. Grammars are also generalized by merging productions that are in some sense similar.

While virtually all approaches classified as grammatical inference use the paradigm of moving from specific to more general models, the opposite strategy is also possible. This involves starting with a completely general model and then modifying it to describe specific rules or patterns. This could be done in the previous example by starting with the finite automaton having a single state and transitions back to that state on input of a and b. If it is then observed that b's always occur in pairs, then a new state can be split off to express this rule. Which paradigm is more appropriate depends on whether, for the specific application, it is more useful in cases of inadequate information to default to allowing anything or to allowing only what has been specifically observed.

Rather than just specifying default behavior in cases of inadequate information, another valid strategy is to assume the availability of a teacher or oracle that can answer the question "Is this string a member of the language?" One of the earliest grammatical inference algorithms used this idea [CM57]. It takes every string in the training set and then asks the teacher, for every substring, whether the strings formed by deleting or repeating this substring are also valid members of the language. This information can then be used to infer a regular grammar. That the number of queries required increases exponentially in the size of the input, however, prevents this approach from being useful. Still, the basic idea of requiring user feedback is

valid for some applications provided a realistic amount of interaction is called for. It is conceivable that some applications may require some form of interaction to produce acceptable results. For example, if the data may contain ambiguities or be composed of multiple components that must be treated in slightly different ways.

2.5 The Inference Process

We have mentioned two possible general paradigms, in Section 2.4, for moving through the infinite space of all languages consistent with a given training set. A specific control strategy is also needed that defines the order of generalization or specialization operations and when the inference process should be terminated. This section discusses the general principles of these strategies.

2.5.1 Non-Probabilistic Grammatical Inference

Most work on grammatical inference has focused on the problem of producing non-stochastic grammars given only positive examples. It is not obvious what kinds of general criteria beyond consistency with the training data can be used to characterize appropriate answers in this situation.

Probably the only generally applicable principle that is easy to define and quantify is that results should be "as simple as possible" in some sense, thus satisfying the principle of Ockham's razor. The additional criteria that have been proposed by researchers to construct practical algorithms are ad hoc. Justifications are often given in the form of specific examples for which the results seem in some sense "intuitively correct." Standard approaches have included limiting results to some characterizable subclass of the target representation, as mentioned in Section 2.3, or making use of a-priori knowledge about the target language to guide heuristic searches. Miclet [Mic90] provides a good overview of these methods, while Angluin and Smith [AS83] discuss the issues and goals associated with inductive inference in general.

2.5.2 Probabilistic Grammatical Inference

Stochastic grammatical inference applications typically have access to only positive examples, however, counts associated with each example are avail-

able. Thus, the input is essentially a statistical sample of an unknown distribution. It can be used either to construct a stochastic model, or as an aid to constructing a non-stochastic model. This situation has several advantages over the non-probabilistic case:

- Inference is more effective since the frequencies represent an additional source of information, effectively taking the place of negative examples as a means to control over-generalization.
- Probabilistic models can be evaluated in a relatively objective manner by comparing the models to the training sets using statistical tests or comparisons based on information theory.
- Probabilistic methods are better able to deal with large, noisy input since unlikely or erroneous examples should have lower frequencies and are therefore less significant.

The fundamental advantage of using frequencies is the ability to objectively evaluate any strings generated by the model but not present in the training set. Such strings are required to have low enough probabilities to be consistent with the assumption that the training set is a random sample of the language generated by the model. Consider the training set $T = \{(a, 20), (aa, 10), (aaa, 5)\}$, consisting of three $(string, frequency)$ pairs for a total frequency of 35. One possible model might be $\{a^n \mid p(a^n) = 0.5^n\}$ which assigns a total probability of 0.125 to strings of four or more a's even though no such strings are present in the sample. A simple statistical test, however, verifies that since T contains only 35 strings it could in fact have realistically been generated by the model. Specifically, a χ^2 test can be used to calculate that the chance of getting T or any less likely sample from the model is somewhere between 0.20 and 0.50 — probably not unlikely enough to conclude that the model is a bad one. For the set $T' = \{(a, 200), (aa, 100), (aaa, 50)\}$, on the other hand, this probability is less than one in a thousand. This reason is that a random sample of 350 strings from the model should have included at least a few longer strings such as *aaaa* or *aaaaa*. Alternative evaluation criteria can be based on things other than strict probability of occurrence. Other approaches include Bayesian comparison to prior probability distributions [SO94] and divergence comparisons based on information theory [SB94].

Since a probabilistic model can be viewed as a non-probabilistic structure augmented with a probability distribution, it is possible to consider

the inference of such a model as comprising two separate components: the learning of the underlying grammar or structure, and the learning of a probability distribution for an already determined structure. The first component is basically just non-probabilistic grammatical inference as discussed in the previous section. The second is sometimes referred to as probabilistic estimation. There are several probabilistic estimation methods that attempt to optimize probability assignments for a fixed structure with respect to a training set. These can be used as simple grammatical inference techniques themselves if the approximate size of the target model is known. Regular languages, for example, can be inferred by first estimating probabilities for a fully connected finite automaton and then deleting the low probability arcs. Better results should be possible, however, by attempting to learn both the structures and the probabilities simultaneously.

Most combined approaches to inferring probabilistic models follow the standard grammatical inference paradigm of generalizing a de-facto model. The strategy used to guide the generalization process is usually heuristic. In addition to heuristics created specifically for this purpose, many non-probabilistic methods have the potential to be adapted to probabilistic analogs.

Chapter 3

Grammatical Inference of Text Structure

3.1 Introduction

The problem of generating a grammar for the structure of a document collection is an application of grammatical inference. Consider the marked up text in Figure 3.1, taken from the *Oxford English Dictionary (OED)* [MBCO33]. A de-facto model can be constructed by first ignoring the content and extracting the tags as shown in Figure 3.2. It can then be represented by the derivation tree, shown in Figure 3.3, where the nodes are labeled with corresponding tag names. The equivalent grammar representation shown in Figure 3.4 includes a production for every non-leaf node along with the root of its subtrees. Frequency information can also be collected by counting how often each production occurs, as shown in Figure 3.5. Each unique non terminal can now be considered separately, by taking all strings associated with a common left-hand side and considering them as a training set for a single sub-problem.

The goal is to generalize these de-facto grammars to reflect the structure of unseen entries, hopefully reconstructing a model close to that used by the original authors in the process. Of course realistically, many more than two example entries will be required to do this effectively.

```
<E><HG><HL><LF>salamandrous</LF><SF>sala&sm.mandrous</SF>
<MF>salama&sd.ndrous </MF></HL><b>,</b> <PS>a.</PS></HG>
<LB>rare</LB><su>-1</su>. <ET>f. <L> L.</L> <CF>
salamandra</CF> <XR><XL>salamander</XL></XR> + <XR>
<XL>-ous</XL></XR>.</ET> <S4><S6><DEF>Living as it were
in fire; fiery, hot, passionate. </DEF><QP><Q><D>1711</D>
<A>G. Cary</A> <W>Phys. Phyl.</W> 29 <T>My Salamandrous
Spirit..my &Ae.tnous burning Humours.</T></Q></QP></S6>
</S4><S4><DEF>So </DEF><SE><BL><LF>salamandry</LF><SF>
sala&sm.mandry</SF> <MF>salama&sd.ndry</MF></BL><DEF>
<PS>a.</PS></DEF></SE><QP><EQ><Q><D>1610</D> <A>Boys</A>
<W>Expos. Dom. Epist. &amp. Gosp.</W> Wks.  (1629) 76
<T>If a Salamandry spirit should traduce that godly labour,
as the silenced Ministers haue wronged our Communion Booke.
</T></Q></EQ></QP></S4></E>

<E><HG><HL><LF>understrife</LF><SF>&sm.understrife</SF>
<MF>u&sd.nderstrife </MF></HL><b>.</b> </HG><LB>poet.</LB>
<ET><XR><XL>under-</XL><HO>1</HO> <SN>5</SN> <SN>b</SN>.
</XR></ET> <S4><S6><DEF>Strife carried on upon the earth.
</DEF><QP><EQ><Q><D>C. 1611</D> <A>Chapman</A> <W>Iliad</W>
xx. 138 <T>We soon shall..send them to heaven, to settle
their abode With equals, flying under&dubh.strifes.</T>
</Q></EQ></QP></S6></S4></E>
```

Figure 3.1: Two marked up dictionary entries.

```
<E>
   <HG><HL><LF></LF><SF></SF><MF></MF></HL>
      <b></b><PS></PS></HG>
   <LB></LB>
   <su></su>
   <ET><L></L><CF></CF><XR><XL></XL></XR><XR>
      <XL></XL></XR></ET>
   <S4><S6><DEF></DEF><QP><Q><D></D><A></A><W></W>
      <T></T></Q></QP></S6></S4>
   <S4><DEF></DEF><SE><BL><LF></LF><SF></SF><MF></MF>
      </BL><DEF><PS></PS></DEF></SE><QP><EQ><Q><D>
      </D><A></A><W></W><T></T></Q></EQ></QP></S4>
</E>

<E>
   <HG><HL><LF></LF><SF></SF><MF></MF></HL><b></b></HG>
   <LB></LB>
   <ET><XR><XL></XL><HO></HO><SN></SN><SN></SN></XR></ET>
   <S4><S6><DEF></DEF><QP><EQ><Q><D></D><A></A><W></W>
      <T></T></Q></EQ></QP></S6></S4>
</E>
```

Figure 3.2: Two entries with the text removed.

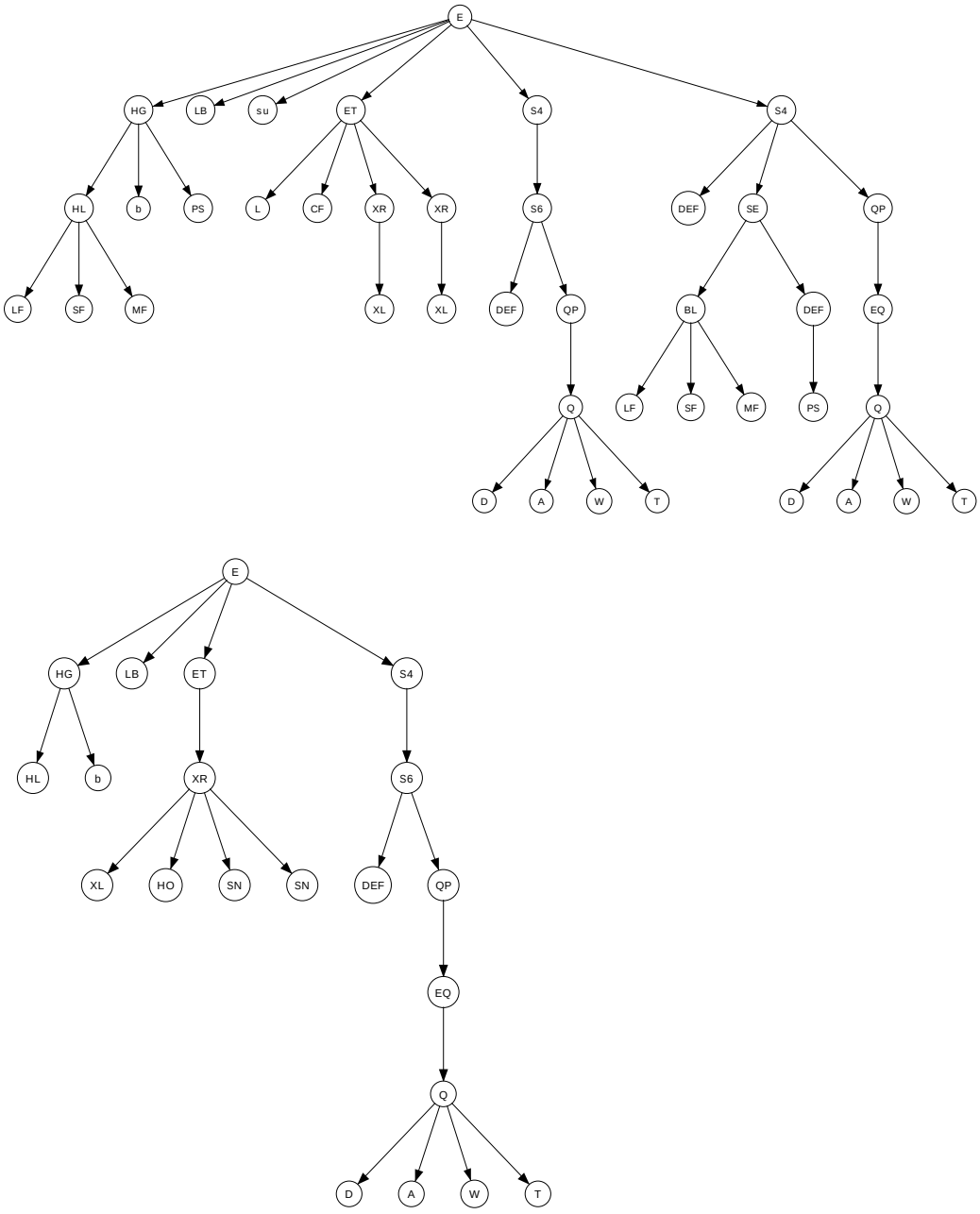

Figure 3.3: The two parse trees.

```
E -> HG LB su ET S4 S4
E -> HG LB ET S4
HG -> HL b PS
HG -> HL b
ET -> L CF XR XR
ET -> XR
S4 -> S6
S4 -> DEF SE QP
HL -> LF SF MF
XR -> XL
XR -> XL HO SN SN
S6 -> DEF QP
DEF ->
DEF -> PS
SE -> BL DEF
QP -> Q
QP -> EQ
BL -> LF SF MF
Q -> D A W T
EQ -> Q
```

Figure 3.4: The de-facto grammar.

```
1 : E -> HG LB su ET S4 S4
1 : E -> HG LB ET S4
1 : HG -> HL b PS
1 : HG -> HL b
1 : ET -> L CF XR XR
1 : ET -> XR
2 : S4 -> S6
1 : S4 -> DEF SE QP
1 : HL -> LF SF MF
2 : XR -> XL
1 : XR -> XL HO SN SN
2 : S6 -> DEF QP
1 : DEF -> PS
1 : SE -> BL DEF
1 : QP -> Q
2 : QP -> EQ
1 : BL -> LF SF MF
3 : Q -> D A W T
1 : EQ -> Q
```

Figure 3.5: The de-facto grammar with production frequencies.

3.2 Existing Work

Three existing approaches to this problem operate by specifying rules that are used to rewrite and combine strings in the training set, while a fourth constructs a de-facto finite automaton which is generalized using state-merges to arrive at a characterizable subclass of the regular languages. None of these approaches take frequency information into account during the inference process.

A research proposal by Chen [Che91] follows the first approach of applying rewrite rules to generalize or simplify productions. The following rule, for example, generalizes a grammar by expanding the language that it accepts:

- $ab^n c \rightarrow ab^+ c$ if n is greater than or equal to a given value

Other rules simplify a language by combining productions:

- $abc \wedge ac \rightarrow ab?c$
- $ab^+ c \wedge ac \rightarrow ab^* c$.

Despite the fact that inference of regular languages can always be reduced to state-merging, Chen justifies the rule rewriting approach by giving the analogy that high level programming languages are useful even though they are all ultimately equivalent in functionality to machine language. No satisfying theoretical framework for this type of approach has ever been developed, and some questions therefore exist. Is there any way to characterize a set of rules that is, in some sense, both complete and orthogonal in functionality? Given that results may depend on which order rules are applied in, can theoretically justifiable rather than heuristic strategies be formulated? How do the effects of rules interact with each other?

Fankhauser and Xu [FX93] describe a similar approach used in a system called *MarkItUp!* that attempts to manage the entire conversion process of marking up a collection of plain documents and constructing a grammar for them. The user is required to manually mark up some examples by delimiting and providing names for elements. The system then tries to determine the context information needed to identify these elements in other documents. An initial grammar is constructed and then incrementally modified as the system comes across new documents that do not fit the current model and asks the user to provide clarifications. The grammar is simplified and generalized as it is constructed by using production rewrite rules in a manner

similar to that described above. An example session given in the article seems to indicate that this approach is workable, although it is not obvious whether it scales effectively to larger inputs.

Shafer [Sha95] describes a C++ library, the GB (Grammar Builder) Engine which produces a grammar by generalizing examples marked up in an SGML-style. The generalization is done using production rewriting in the same manner as the two systems mentioned above, although the set of rewrite rules is slightly different. Shafer does not report what kind of results are obtained. It is implied, however, that the system has been used on some real-world document collections.

Work by Ahonen, Mannila and Nikunen [AMN94b, AMN94a] uses a more classical grammatical inference approach based on the notion of a $(k-h)$ contextual language, an extension that they propose to the idea of k-contextuality. A language is $(k-h)$ contextual if any two identical substrings of length k that appear in separate example strings or different places in the same example string are guaranteed to pass through the same last h states of a recognizing automaton. With this definition, a de-facto finite automaton is first constructed from the examples and then modified with state merges to produce the minimum $(k-h)$ contextual language for chosen values of k and h. The algorithm basically looks for length k paths through the automaton that are labeled with the same strings, and merges their last h states. This can be formulated equivalently using n-grams rather than a finite automaton to construct a more efficient implementation. In either case, the final result is then converted into a grammar. The approach is augmented with some basic interactive operations that allow the user to view example strings contained in the final language and delete those judged to be incorrect (as long as their removal doesn't prevent the language from remaining $(k-h)$ contextual). Frequency information is also given limited consideration by discarding examples less frequent than an arbitrarily chosen threshold. Experimental results show that the final grammar for complex documents can be excessively large and unwieldy, requiring significant manual rewriting to extract common subexpressions before being understandable.

3.3 Proposed Approach

It is the premise of this thesis that stochastic grammatical inference approaches are better suited to the application domain of text structure than

the non-stochastic ones that have been used to date. The primary gain from using a stochastic approach is expected to be in the effectiveness of the inference process. Realistic document collections with complex structure are likely to be large and to contain errors or pathological exceptions. Both these characteristics are known to be better addressed by stochastic approaches.

Chapter 4

Inference Method

4.1 Introduction

This chapter details the stochastic inference algorithm we have chosen to adopt (Section 4.2), and the modification made to it (Section 4.3). Factors that influenced the original choice included the following:

- Stochastic finite automata were judged to be an appropriate target representation for modeling text structure.
- An inference paradigm that generalized from a specific de-facto model seemed appropriate for application to text structure.
- The algorithm produces a stochastic model directly rather than using separate grammatical inference and probabilistic estimation components.
- The algorithm is statistically justifiable rather than relying on arbitrary heuristics.

Even though the choice appeared to be a good one based on these characteristics, there are no overwhelming reasons to suppose that other algorithms might not have turned out to be better. Investigation of alternative approaches is left for future work.

4.2 Basic Method

This section gives a complete description of the algorithm ALERGIA proposed by Carrasco and Oncina [CO94b]. ALERGIA is itself an adaptation of a non-stochastic method described by Oncina and García [Oa92].

4.2.1 Target Representation

The algorithm generates deterministic stochastic finite automata. Formally, a stochastic finite automaton or SFA is defined as follows: Let $\mathcal{A}$ be a finite alphabet, and $\mathcal{A}^*$ be the set of all strings over $\mathcal{A}$. A stochastic finite automaton, $A = (\mathcal{A}, Q, P, q_0)$, is defined by an alphabet $\mathcal{A}$, a finite set of nodes $Q = \{q_0, q_1, \ldots, q_n\}$, with q_0 the initial node, and a set P of probability matrices $p_{ij}(a)$ giving the probability of a transition from node q_i to node q_j on the symbol $a \in \mathcal{A}$. $p_{i\mathbf{f}}$ is the probability that a string terminates at node q_i. (This does not denote a transition to some separate, final state. Any node i with a non-zero value of $p_{i\mathbf{f}}$ is a final state.)

For a deterministic stochastic finite automaton or DSFA, a given state q_i and symbol a has at most one other node q_j such that $p_{ij}(a) > 0$. In this case, we can unambiguously denote the probability for a transition as $p_i(a)$, and define a transition function $\delta(q_i, a) = q_j$ that returns the unique destination node. The probability $p(w)$ for the string $w = x_1x_2x_3\ldots$ to be generated by a DSFA is defined as:

$$
\begin{aligned}
p(w) &= p_0(w) \\
\forall i,\ 0 \leq i \leq |Q| & \\
p_i(w) &= \begin{cases} p_i(x_1)p_{\delta(q_i,x_1)}(x_2x_3\ldots), & |w| \geq 1 \\ p_{i\mathbf{f}}, & |w| = 0 \end{cases}
\end{aligned}
$$

The language generated by an automaton A is defined as:

$$L = \{w \in \mathcal{A}^* \mid p(w) \neq 0\}$$

An SFA should be *consistent*, i.e., the probability of all strings in the language should add up to 1. This is ensured if the following constraint holds for every node q_i:

$$p_{i\mathbf{f}} + \sum_{q_j \in Q} \sum_{a \in \mathcal{A}} p_{ij}(a) = 1$$

Note that deterministic and non-deterministic stochastic finite automata are equivalently expressive. See the book by Fu [Fu82] for a more complete description of the properties of stochastic automata.

4.2.2 Inference Paradigm

Generalization is performed on the de-facto model using a state-merging paradigm. The primitive operation of merging two states replaces them with a single state, labeled by convention with the smaller of the two state identifiers. All incoming and outgoing transitions are combined and the frequencies associated with any transitions they have in common are added, as are the incoming and termination frequencies. This is related to the merging operation used in the algorithm for minimizing a non-stochastic finite automaton [HU79]. The difference is that merges are allowed that change the accepted language.

State merging paradigms rely on a non-stochastic theoretical framework: given a language L, the minimum DFA generating L is called the *canonical acceptor* $C(L)$. Let T be the de-facto deterministic finite automaton or prefix tree for a finite sample S. If π is a partition of the set Q of nodes of T, $\pi(T)$ is the automaton obtained by merging all nodes in each block of π. It is well known that, for a sample S that covers every transition of the canonical acceptor, there exists a partition π such that $\pi(T)$ is exactly $C(L)$. Therefore, the problem of identifying L is reduced to finding the partition π. This can be accomplished by testing nodes for equivalence and merging them two at a time. An algorithm must define two things: how to test the equivalence of two nodes, and the order in which nodes should be tested.

4.2.3 Equivalence Criterion

Two nodes are defined to be equivalent if their outgoing probabilities are the same for every symbol $a \in \mathcal{A}$ and their destination nodes are also equivalent:

$$q_i \equiv q_j \iff \forall a \in \mathcal{A} \begin{cases} p_i(a) = p_j(a) \\ p_{i\mathbf{f}} = p_{j\mathbf{f}} \\ \delta(q_i, a) \equiv \delta(q_j, a) \end{cases}$$

Therefore, two nodes are compared by checking the equivalence of their termination probabilities and each of their transition probabilities, followed by

a recursive check of any destination nodes reachable via transition symbols they have in common.

The equivalence of two transition or termination probabilities is checked using a statistical test. The test used in ALERGIA will be presented here in a more standard framework than given by Carrasco and Oncina so that later modifications can be seen to fit in more naturally. Elementary statistical hypothesis tests (and scientific tests in general) follow a standard procedure [Chr92]:

1. Formulate a null and alternative hypothesis (H_o and H_a) concerning the parameter of interest.

2. Design an experiment that a "reasonable" person would regard as providing proof concerning the truth of the alternative hypothesis. This includes choosing a significance level α that will be used to make the final conclusion.

3. Decide on a meaningful test statistic chosen to model the distribution of the parameter of interest.

4. Conduct the experiment and compute the value of the test statistic. Calculate the probability of getting this value or any value more extreme assuming the null hypothesis H_o is true. This probability is often called the p-value for the test.

5. If the p-value is small ($p \leq \alpha$) it signifies that the probability of getting that particular sample statistic is remote when the null hypothesis is true. Therefore, "reasonable" persons would reject the null hypothesis in favor of the alternative. (What value of α can be considered "small" is a matter of choice, but common practice uses the guideline of letting 0.05 or less signify that condition.) Note that a small p-value does not necessarily mean the null hypothesis is false. It may indicate a problem with how the data was obtained, or that the model chosen for the distribution of the parameter was inappropriate, or that the null hypothesis is true and a very unlikely outcome has occurred. If however, we assume that (1) the model is correct, and (2) the data is valid, then a small p-value logically calls the validity of the null hypothesis into question.

The test used in ALERGIA for checking whether two transition or termination probabilities p_1 and p_2 are equal, can now be presented as follows:

1. The null and alternative hypotheses are

   $H_o : p_1 = p_2$ (the two probabilities are the same)
   $H_a : p_1 \neq p_2$ (the two probabilities are different)

2. Let n_1, n_2 be the number of strings that arrive at the two nodes. Let f_1, f_2 be the number of strings that arrive and then satisfy the event of interest (either termination at the current node or transition on a given symbol). Let α be a significance level chosen as the cutoff probability below which H_o will be rejected.

3. p_1 and p_2 represent the unknown true probabilities and will be experimentally estimated by f_1/n_1 and f_2/n_2 respectively. It is assumed that these can be modeled by a *binomial distribution*. (A binomial distribution $f(x, n, p)$ gives the probability of getting x positive outcomes when taking n samples from an infinite population where a positive outcome has probability p and a negative outcome has probability $1 - p$.) The test statistic TS will be the absolute difference of the observed proportions:

$$TS = \left| \frac{f_1}{n_1} - \frac{f_2}{n_2} \right|.$$

4. We now wish to calculate the probability of a given value of TS assuming H_o is true (the unknown true probabilities p_1 and p_2 are equal). Because exact calculations involving binomial distributions are expensive, the following approximation is used. The Hoeffding bound [Hoe63], states that for a binomial distribution with probability p and observed frequency f out of n tries

$$\left| p - \frac{f}{n} \right| < \sqrt{\frac{1}{2n} \log \frac{2}{\alpha}}$$

   with probability larger than $(1 - \alpha)$. For two binomial distributions with equal probabilities, the two inequalities can be added giving

$$TS < RHS = \sqrt{\frac{1}{2} \log \frac{2}{2\alpha}} \left(\frac{1}{\sqrt{n_1}} + \frac{1}{\sqrt{n_2}} \right)$$

   with probability larger than $(1 - \alpha)$. Although this does not allow the exact calculation of the p-value, we can conclude that it is less than α if TS is experimentally observed to be greater than RHS.

5. If, based on the above decision rule, we conclude that an observed value of TS has a p-value less than α, the implication is that the observed frequencies were very unlikely to have come from the same distribution. We therefore reject the null hypothesis and assume the two probabilities are different. Otherwise, the null hypothesis stands and we assume the two probabilities are equivalent.

4.2.4 Testing Order

The final detail that must be mentioned before giving the algorithm is the order in which nodes are compared. This is defined by numbering nodes, starting at 0, in a lexical order based on the string prefixes that arrive at those nodes. Pairs of nodes (q_i, q_j) are then tested by varying j from 1 to the number of nodes, and i from 0 to $j-1$.

The order in which nodes are tested and merged can have a fairly arbitrary effect on the final result. This is because the merging operation replaces multiple nodes with a single representative which may not test equal to all of the nodes that the original components would have. This is basically the problem of *reversals* seen in the task of clustering points or vectors when centroid or median representatives are used [And73].

Whether the testing order described has any real significance is unclear. For a non-stochastic version of the algorithm, the ordering is necessary to prove the ability to identify languages in the limit. For the stochastic version, it only ensures that node comparisons will take place roughly in order of entering frequency, since nodes with lower indices, being closer to the root of the tree, will usually have higher frequencies.

4.2.5 Algorithm

Figure 4.1 is the main pseudo-code for ALERGIA. n_i, n_j denote the entering frequencies of nodes i and j; $f_i(a)$, $f_j(a)$ denote the transition frequencies on input of symbol a from nodes i and j; and, $f_{i\mathbf{f}}$, $f_{j\mathbf{f}}$ denote the termination frequences at nodes i and j. Figures 4.2, 4.3 and 4.4 are the supporting subroutines.

Note that the algorithm guarantees a deterministic output by merging any destination nodes reached via transitions that leave the same state but have a different label. Nodes merged in this way will always have been found to be equivalent since the indeterminism in their common parent will have

```
algorithm ALERGIA
input:
        S : sample set of strings
        α : significance level
output:
        stochastic deterministic finite automaton
begin
        A = stochastic prefix tree acceptor from S
        for j = successor(firstnode(A)) to lastnode(A)
                for i = firstnode(A) to j
                        if compatible(i, j)
                                merge nodes i and j
                                determinize(A, i)
                                exit (i-loop)
                        end if
                end for
        end for
        return A
end algorithm
```

Figure 4.1: Algorithm **ALERGIA**

```
algorithm compatible
input:
        i, j : nodes
output:
        boolean
begin
        if different( n_i, f_if, n_j, f_jf )
                return false
        end if
        for (∀a ∈ A)
                if different( n_i, f_i(a), n_j, f_j(a) )
                        return false
                end if
                if not compatible( δ(i, a), δ(j, a))
                        return false
                end if
        end for
        return true
end algorithm
```

Figure 4.2: Algorithm **compatible** recursively checks $q_i \equiv q_j$.

algorithm **determinize**
input:
A : finite automaton
i : node in A, possibly with duplicate arcs
output:
equivalent deterministic finite automaton
begin
do $(\forall a \in \mathcal{A})$
if (node i has two transitions $\delta_1(i,a)$ and $\delta_2(i,a)$ on a)
merge nodes $\delta_1(i,a)$ and $\delta_2(i,a)$
determinize($A, \delta_1(i,a)$)
end if
end do
end algorithm

Figure 4.3: Algorithm **determinize** eliminates nondeterminism by merging nodes.

algorithm **different**
input:
n_1, n_2 : number of strings arriving at each node
f_1, f_2 : number of strings ending/following a given transition
output:
boolean
begin
return $\left|\frac{f_1}{n_1} - \frac{f_2}{n_2}\right| > \sqrt{\frac{1}{2}\log\frac{1}{\alpha}}\left(\frac{1}{\sqrt{n_1}} + \frac{1}{\sqrt{n_2}}\right)$
end algorithm

Figure 4.4: Algorithm **different** checks if the difference between two observed proportions is statistically significant.

been created by merging two nodes found to be equivalent using the recursive test.

The worst case time complexity of this algorithm is $O(n^3)$. This corresponds to an input where no merges take place, thus requiring that all $n(n-1)/2$ pairs of nodes be compared; and furthermore, where the average recursive comparison continues to $O(n)$ depth. In practice, the expected running time is likely to be much less than this. Carrasco and Oncina report that, experimentally, the time increases linearly with the number of strings in the training set, as generated by a fixed size model. A better test might have varied the size of the target model. This could be expected to affect the time quadratically.

Carrasco and Oncina give a proof that the global probability of the algorithm making an error asymptotically approaches zero as the size of the training set increases relative to the size of the target model. They do not, however, characterize the speed of this convergence, nor do they consider the effect of errors in the data. The problems and modifications discussed in the next section essentially address the problem of applying the algorithm to real data whose adequacy must be checked rather than assumed.

4.3 Modifications

Problems with the original algorithm were identified when applying it to the *OED* data. This section describes these problems, their interpretation, and the resulting modifications that were made to the method.

4.3.1 Low Frequency Components

A serious problem resulted from how the algorithm treats nodes having very low entering frequencies (this was initially only noticed for nodes with entering frequencies of 1). Such nodes would tend to get merged inappropriately. Specifically, they always tested equal to the first node with which they were compared.

The mechanical explanation for this can be seen by looking at the equations on page 32. If either n_1 or n_2 is small enough, the value of RHS can be greater than one. Since observed proportions cannot differ by more than one, nodes with low frequencies will tend to test equal to any other.

Low frequency nodes can be interpreted as being components of the data that are impossible to distinguish with the required confidence (as specified by α). This essentially means that insufficient information is available to apply a statistical test. Such nodes may correspond to errors in the data or just to unlikely strings.

The overall problem is that the algorithm defaults to merging nodes (accepting the null hypothesis) in cases of inadequate information. This damages the resulting structure (although a strictly probabilistic evaluation may not catch the inaccuracy since the bad transitions have relatively low probabilities). Note that this behavior is somewhat worsened by the fact that the Hoeffding bound used to construct the test is not a very tight inequality. The fundamental problem would remain, however, even if the required probability values were calculated exactly.

Some way is needed to separate the adequate and inadequate components of the training data. Cases where RHS will be greater than one and merging will therefore be automatic are easy to single out as inadequate. It must also be assumed, however, that borderline cases will exist for which incorrect merges may not be automatic but will be unacceptably likely. Ideally, it would be desirable to have a testing framework that better characterizes the relevant interactions. The criterion for identifying borderline cases could then be formulated in some statistically justifiable way.

Statistical experiments often require sample sizes to be considered, and therefore, the relevant relationships are well understood. A hypothesis test can make two types of errors. The null hypothesis can be incorrectly rejected when it is in fact true, or it can be incorrectly accepted when it is in fact false. The significance level α used in most statistical tests is a bound on the chance of the first type of error. Another bound, usually called β, can also be be made on the second type of error ($1 - \beta$ is sometimes referred to as the *power* of the test, i.e., its chance of being able to correctly reject the null hypothesis). The relationship between the four possible outcomes of a statistical test and these two parameters is shown in Figure 4.5.

α and β are closely related to sample size. Specifically, any two of these are free parameters from which the third is determined. For the basic test used in ALERGIA, the sample sizes were predetermined by the data and α was chosen by the user. β, the chance of incorrectly accepting H_o and concluding that two nodes were equivalent, was therefore implied, and could vary arbitrarily depending on the sample size. It could, for low enough sample sizes, increase to the point where incorrect merging was automatic.

		DECISION	
		Reject H_o	Accept H_o
ACTUAL SITUATION	H_o is true	Type I Error α	Correct Decision $1-\alpha$
	H_a is true	Correct Decision $1-\beta$	Type II Error β

Figure 4.5: The four possible outcomes of a statistical test and the parameters associated with their probabilities.

It is now clear that there are three ways of choosing the test parameters:

- α can be chosen and β can be allowed to vary arbitrarily. Therefore, the default behavior for cases of insufficient information (i.e. too small sample sizes) will be to merge nodes.
- β can be chosen and α can be allowed to vary arbitrarily. The default behavior will be to leave nodes separate.
- both α and β can be chosen. Nodes where the sample size prevent these bounds from being satisfied can be flagged for alternative treatment.

It is the third option that fulfills our goal of providing a method for separating the input into adequate and inadequate components in a statistically justifiable way.

We now detail an alternative test that explicitly relates all the parameters of interest. Assume as before that p_1, p_2 represent the unknown, true probabilities and that f_1/n_1, f_2/n_2 will serve as the estimates. The following transformation can be used to map the difference of two binomial proportions to a distribution that is approximately normally distributed with a mean of zero and variance of one [Sil49]:

$$t = \frac{\left| 2 \arcsin \sqrt{f_1/n_1} - 2 \arcsin \sqrt{f_2/n_2} \right|}{\sqrt{1/n_1 + 1/n_2}}$$

This transformation is used because quantiles of the standard normal distribution can easily be calculated. Therefore, t can be used as a test statistic, and a p-value can be obtained. Let z_x represent the t value at which the

cumulative probability of a standard normal distribution is equal to $1 - x$. The p-value for a two-sided test will be greater than $1 - \alpha$ if $t > z_{\alpha/2}$.

The probability of incorrectly accepting H_o cannot be bounded by specifying only a single β parameter. This is because that probability is dependent on the unknown true difference between p_1 and p_2. (A very small true difference implies a high likelihood of incorrectly concluding the proportions are equal whereas a large true difference makes this unlikely.) Therefore, it is not possible to guarantee that H_o be correctly rejected with probability $1-\beta$ in general, but it is possible to make this guarantee provided p_1 and p_2 differ by more than some specified difference ϵ. The power requirement can now be stated as follows:

$$P(t > z_{\alpha/2} \mid p_1 - p_2 \geq \epsilon) \quad < \quad 1 - \beta.$$

This requires that the sample sizes satisfy

$$\frac{1}{n_1} + \frac{1}{n_2} < \left\{ \frac{2\epsilon^*}{z_{\alpha/2} + z_\beta} \right\}^2$$

where $\epsilon^* = 2 \arcsin \sqrt{p_1} - 2 \arcsin \sqrt{p_2}$. Unfortunately, ϵ^* depends on the positions of p_1 and p_2 in the unit interval and not just their absolute difference. In the absence of known constraints on their positions, it is standard practice to use the minimum possible value (which will somewhat overestimate the required sample sizes):

$$\epsilon^*_{min} = \arcsin \sqrt{0.50 + \epsilon/2} - arcsin \sqrt{0.50 - \epsilon/2}.$$

For experimental design, the next step would be to solve for the required sample sizes for chosen values of α, β and ϵ. For our purposes, sample sizes cannot be chosen, and it is therefore enough that we now have equations relating all of the parameters.

The new test can now be integrated into the algorithm. Nodes that are too infrequent to satisfy the global probability bounds specified by α, β and ϵ can be marked as low frequency components and left for some other treatment. The specific procedure that we have chosen to accomplish this is shown in the updated algorithm in Figure 4.6. Algorithm **bigEnough** in Figure 4.8 checks whether two node frequencies n_1 and n_2 allow the global probability bounds to be satisfied. The updated Algorithm **different** in Figure 4.7 implements the new decision test. The procedures **compatible**

and **determinize** remain unchanged. In the main procedure, an boolean size flag is associated with every node and initially set to false. A size flag is set to true as soon as the node is involved in a successful call to **bigEnough**. Low frequency components are those that still have their size flags set to false when the algorithm terminates.

Note that there are circumstances where low frequency nodes can be compared, and will be merged if they test equal. This can happen since the check for adequate size is only done for the initial call to **compatible**. Recursively called comparisons are allowed to take place even if the nodes would have been classified as low frequency. When this happens, and the nodes test equivalent, the ancestor nodes at the top of the recursion can also test equivalent and merge. The low frequency nodes are then merged in the determinization phase.

The next step after the algorithm has run its course is to do something with the remaining low frequency components. Possible approaches are:

- Assume the most specific possible model by leaving the components separate. This strategy is the same as leaving β fixed and allowing α to grow arbitrarily high.

- Assume the most general possible model by merging every low frequency node with its immediate parent. This is not the same as fixing α and letting β grow arbitrarily high, since the low frequency nodes are merged into specific places rather than with the first nodes to which they are compared. The result of merging all low frequency nodes with their immediate parents is that any terminals occurring in a low frequency subtree are allowed to occur any number of times and in any order starting at the closest high-frequency parent.

- Merge low frequency nodes into parts of the automaton based on other criteria.

Many possible criteria can be invented for the third approach. Note, however, that statistical approaches or ones that make use of frequency information are unlikely to work in general, since it has already been determined that the relevant components have insufficient frequency information for meaningful statistical comparison. The situation is therefore basically a reversion to non-probabilistic inference, except that we have at our disposal a partial model into which we can try to fit the low frequency cases.

```
algorithm updated-ALERGIA
input:
        S : sample set of strings
        α : 1-confidence level
output:
        stochastic deterministic finite automaton
begin
        A = stochastic prefix tree acceptor from S
        ∀i  sizeFlag(i) = false
        for j = successor(firstnode(A)) to lastnode (A)
                for i = firstnode(A) to j
                        if bigEnough(n_i, n_j)
                                sizeFlag(i) = true
                                sizeFlag(j) = true
                                if compatible(i, j)
                                        merge nodes i and j
                                        determinize(A, i)
                                        exit (i-loop)
                                end if
                        end if
                end for
        end for
        return A
end algorithm
```

Figure 4.6: Algorithm **updated-ALERGIA**

algorithm **different**
input:
n_1, n_2 : number of strings arriving at each node
f_1, f_2 : number of strings ending/following a given transition
output:
boolean
begin
return $\frac{\left| 2 \arcsin \sqrt{f_1/n_1} - 2 \arcsin \sqrt{f_2/n_2} \right|}{\sqrt{1/n_1 + 1/n_2}} > z_{\alpha/2}$
end algorithm

Figure 4.7: Algorithm **different** checks if the difference between two observed proportions is statistically significant using a transformation to a unit normal distribution.

algorithm **bigEnough**
input:
n_1, n_2 : number of strings arriving at each node
output:
boolean
begin
return $\frac{1}{n_1} + \frac{1}{n_2} < \left\{ \frac{2\epsilon^*_{min}}{z_{\alpha/2} + z_\beta} \right\}^2$
end algorithm

Figure 4.8: Algorithm **bigEnough** checks whether two sample sizes are large enough to satisfy the probability bounds specified by α and β.

One simple approach based on this observation is to locate a position in the high frequency model from which an entire low-frequency subtree can be parsed. This subtree can then be merged into the rest of the model by replacing it with single transitions to the identified position. If more than one possible position exists, the merge can be treated as tentative and adjusted in a later interactive stage.

In fact, the idea of *tentative merging* can be used as the basis of a general strategy. An ordered list of heuristics such as the one just mentioned can be defined. All low frequency components can then be merged into positions in the model determined by the first heuristic in the list. If a problem is later identified with a particular resulting *tentative transition* then the subtree can instead be merged into a position determined by the next heuristic in the list.

4.3.2 Generalization versus Reconstruction

The algorithm as described so far is appropriate for reconstructing exact models. This is the task usually used to test stochastic grammatical inference algorithms. It is less suited for the task of generalizing real data, which may not behave as if generated by an exact model.

For reconstruction, it is only appropriate to generalize to the point where statistically insignificant features of the data are discarded. This is what the algorithm is doing when it concludes that nodes with small differences could have come from identical sources. For data not generated by an exact model, a good answer may necessarily generalize to the point where features of the data that are definitely statistically significant are neglected.

The level of generalization determines how significant the features that can be neglected are allowed to be. Ideally, this should be adjustable so that the size of the final model can effectively be chosen to vary smoothly from a single state all the way up to the unmodified de-facto model.

Note that the overall purpose of generalization in learning tasks is to avoid models that are over-fitted to incidental features of the specific training data being used. This is done so that models are more likely to be applicable to data outside of the training sets used to construct them. The distinction that we have pointed out is that over-fitting when reconstructing exact models can be easily avoided by just ignoring statistically insignificant components of the data; for data not generated by an exact model, the level of generalization should be adjustable and may have to be chosen through experimentation.

Generalization can be increased in the algorithm in its current form by

adjusting the equivalence test so that larger differences between proportions are treated as insignificant. This can be done by decreasing the value of α or increasing the value of β, both of which widen the confidence intervals. This, however, creates a problem due to the additional effect that these parameters have on the sample size requirements. Specifically, the cutoff point between high and low frequency components will increase quadratically with the size of the confidence intervals.

An alternative approach that does not affect required sample sizes is to change the hypotheses for the statistical tests. Rather than testing whether two proportions can plausibly be equal, test whether they can plausibly differ by less than some specified parameter γ:

$$H_o : |p_1 - p_2| \leq \gamma$$
$$H_a : |p_1 - p_2| > \gamma.$$

This can be done by simply subtracting γ from the observed difference in the proportions and then checking for equality. Note that the transformation applied to proportions must be taken into account when incorporating this into the described test. This is shown in the modified **different** algorithm in Figure 4.9.

4.4 Summary of Parameters

A drawback of the modified algorithm is that the number of parameters has been increased from one to four.

- γ is the maximum difference in true proportions for which the algorithm should merge two states.
- α is the probability bound on the chance of making a type I error (incorrectly concluding that the two proportions differ by more than γ)
- β is the probability bound on the chance of making a type II error (incorrectly concluding that the two proportions differ by less than γ) when the true difference in the proportions is at least $\gamma + \epsilon$ (ϵ being the fourth parameter)

Being able to adjust these parameters to correct for specific problems requires an understanding of their practical effects.

algorithm **different**
input:
n_1, n_2 : number of strings arriving at each node
f_1, f_2 : number of strings ending/following a given transition
output:
boolean
begin
$\pi_1 \leftarrow f_1/n_1$
$\pi_2 \leftarrow f_2/n_2$
if $|\pi_1 - 0.5| < |\pi_2 - 0.5|$
$\pi_1 \leftarrow \pi_1 + \gamma \cdot \text{sign}(\pi_2 - \pi_1)$
else
$\pi_2 \leftarrow \pi_2 + \gamma \cdot \text{sign}(\pi_1 - \pi_2)$
end if
return $\frac{|2 \arcsin\sqrt{\pi_1} - 2 \arcsin\sqrt{\pi_2}|}{\sqrt{1/n_1 + 1/n_2}} > z_{\alpha/2}$
end algorithm

Figure 4.9: Algorithm **different** checks if the difference between two observed proportions could realistically be less than γ.

Choosing γ controls the amount of generalization that will take place. Setting it to 0 may result in very few states being merged whereas setting it to 1 will always result in an output with a single state (effectively a 0-context average of the frequency of occurrence of every symbol).

Changes to α and β affect two things. Increasing these values will change the width of confidence intervals and increase the cutoff point between high and low frequency components. Direct control over the second characteristic is more likely to be of interest for the purpose of interactive adjustment. If it is observed that too many nodes have been classified as low frequency then these parameters should be increased. For the sake of simplification, it is possible to always have both of these values equal and adjust them together as a single parameter. This does not seriously reduce the level of useful control over the algorithm's behavior.

The ϵ parameter determines what difference the β probability applies to. This must be specified somewhere but is not an especially useful value over which to have control. It should probably be fixed or tied to the size of the input and the value of γ. If the second option is used, possible considerations include the fact that ϵ should not be less than the smallest possible difference in proportions as determined by the two largest states, and the observation that a value larger than $1 - \gamma$ is not meaningful.

Overall then, it can be seen that control is only really needed over two major aspects of the inference process. Choosing a combined value for α and β effectively sets the cutoff point between the significant data and the low frequency components. Choosing the value of γ effectively controls the amount of generalization. Both of these adjustments will be demonstrated in Chapter 6.

Chapter 5

Understanding Stochastic Automata

5.1 Introduction

Grammars for text structure serve two purposes: they support automatic operations such as data validation, and they provide general understanding of the text. The understandability of a grammar is, to some extent, a characteristic of the grammar itself, dependent on its simplicity, representation and organization. It is also related, however, to what techniques are available to support understanding. This chapter is concerned with techniques applicable to stochastic finite automata. Three approaches will be presented: summarization using metrics (Section 5.2); visualization using graphs (Section 5.3); and generation of characteristic samples (Section 5.4).

While operations such as editing and query formulation require understanding of the final form of a grammar, intermediate models generated during the inference process should also be understood to allow interactive evaluation and feedback. The algorithm described simply produces a model using a given set of parameters. We therefore assume that the overall inference will be conducted as an iterative process of generating a model using a starting set of parameters, understanding the model, identifying problems, and making adjustments.

5.2 Metrics

An elementary understanding can be gained about a model by examining summary values or metrics. For stochastic finite automata, these can be simple counts, such as the number of states and edges, or other values, such as the maximum depth or the number of cycles. These can give an overall idea of an automaton's characteristics but are not generally useful for pinpointing specific problems.

Single value metrics can also be designed to quantify how well a grammar fits a given training set. Unfortunately, such metrics usually have no directly understandable interpretation. They are therefore typically used to support automatic rather than interactive evaluation by guiding heuristic searches. For example, the χ^2 value, which is used to support a statistical test that compares multinomial distributions, is calculated as

$$\sum_{s \in T} \frac{(f_M(s) - f_T(s))^2}{f_M(s)},$$

where T is the training set, $f_T(s)$ is the frequency of s in T and $f_M(s)$ is the frequency of s predicted by the model for a training set as big as T. This metric is used to guide a grammatical inference method described by Maryanski and Booth [MB77]. An information theory metric, the *divergence* of a model with respect to the training set can be estimated by

$$\sum_{s \in T} P_T(s) \log \frac{P_T(s)}{P_M(s)},$$

where $P_T(s)$ and $P_M(s)$ are the probabilities of s in the training set and the model respectively. This is used as the evaluation criterion in experiments described by by Sánchez and Benedí[SB94]. Another metric depends on the notion of a *prior probability* — essentially a predetermined constraint on the distribution of the desired language. A model M assigns a probability $P(T|M)$ to a given data set T, and independent prior probability distributions $P(M)$ and $P(T)$ can be defined for the model and the training set. For instance, an inverse relationship could be defined between $P(M)$ and the size of M. Bayes' rule can then be used to calculate $P(M|T)$ (literally, the likelihood of the model given the data) as follows:

$$P(M|T) = \frac{P(M)P(T|M)}{P(T)}.$$

Using a Bayesian metric to guide language learning was first proposed by Horning [Hor69]. It has more recently been applied in this capacity by Stolcke and Omohundro [SO94].

5.3 Graphs

Automata can be directly visualized as bubble diagrams. The two main obstacles to this are the general difficulty of finding a good layout for a given graph, and the possibility that a graph may be too large to be represented in a single diagram. These are basically problems of graph visualization, a subject that is a field in its own right. The first task of basic graph layout involves node placement and edge crossover minimization and was performed for the examples in this thesis using the graph visualization program *daVinci* [FW95]. The second problem of visualizing a large graph, must be addressed by strategies that display only part of the graph at one time. Possible approaches include pruning components below a frequency threshold, displaying only subtrees reachable from a given prefix, or collapsing subgraphs into single nodes.

5.4 Representative Strings and Expressions

Another way to understand a model is to list a finite sample of structures that somehow characterize the generated language. These structures can be single strings, or expressions that represent sets of equivalent strings. Strings can be defined as equivalent if they follow similar paths through the graph. For instance, all paths covering exactly the same subset of transitions could be defined as equivalent.

To gain general understanding of a model, strings or expressions can be listed in order of probability, thus indicating typical representatives of the language. This can be accomplished by performing a traversal of the graph using a queue prioritized by partial string or path probabilities.

To pinpoint specific problems, it would be better to list in reverse probability order, thus focusing on low probability strings which are more likely to represent errors. Unfortunately, there is no clear way to do this. Strategies can be devised, however, to list in approximate reverse orders.

Feedback based on strings or expressions is straightforward: either a

string or expression is appropriate to include in the language or its inclusion is an error. For an error, there are two possible adjustments that can be made to the model. If the string or expression crosses a tentative transition then the transition can be moved. Otherwise, parameters must be adjusted and the inference algorithm re-applied. Both of these adjustments will be illustrated in the next chapter.

The manual task of recognizing erroneous strings or expressions can be facilitated by comparing the probabilities predicted by the model and those implied by the training set. How significantly these probabilities differ for a given string can be measured using any of several comparison values. Absolute difference is one choice, but this does not consider that given differences are more significant for smaller probabilities. 0.011 and 0.001, for example, differ more significantly than 0.80 and 0.81 even though their absolute differences are the same. Two comparisons that compensate for this are the contributions of individual strings to the χ^2 and divergence values mentioned in Section 5.2.

In summary, automata can be understood by listing finite samples of characteristic strings or expressions. For general understanding, listing is best done in probability order. For pinpointing problems, listing can be done in some approximate reverse probability order. Once generated, the sample can be sorted according to various metrics that compare the predicted and observed probabilities. The model can then be adjusted to remove any strings or paths that are judged to be errors.

Chapter 6

Experimental Results

6.1 Introduction

This chapter presents experimental results obtained using a C++ implementation of the algorithm. Section 6.2 gives a small example that illustrates interactive feedback. Section 6.3 looks at two larger examples to demonstrate basic techniques for general understanding. Section 6.4 discusses how well stochastic finite automata were found to model text structure.

6.2 Feedback Example

This section looks at the pseudo-quotation paragraph or PQP element from the *OED* data [1]. The ninety strings representing that element's examples are shown in Figure 6.1 and the corresponding prefix tree, which has a 119 nodes, is shown in Figure 6.2. Nodes in the diagram are marked with their id numbers, followed in square brackets by their entering frequencies and their termination probabilities. Arcs are marked with transition symbols, followed in square brackets by their transition probabilities.

The running time for a single application of the algorithm to this data was around one second on a Sun 4. An initial inference result with $\alpha = \beta = \gamma = 0.025$ and $\epsilon = 0.1$ is shown in Figure 6.3. Low frequency nodes in that diagram are represented as rectangles.

[1] See the book by Berg [Ber93] for an explanation of the structural elements used in the *OED*.

```
21:
524:    EQ
5:      EQ,LQ
294:    EQ,Q
1:      EQ,Q,LQ
156:    EQ,Q,Q
64:     EQ,Q,Q,Q
30:     EQ,Q,Q,Q,Q
1:      EQ,Q,Q,Q,Q,LQ
15:     EQ,Q,Q,Q,Q,Q
8:      EQ,Q,Q,Q,Q,Q,Q
5:      EQ,Q,Q,Q,Q,Q,Q,Q
3:      EQ,Q,Q,Q,Q,Q,Q,Q,Q
3:      EQ,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      EQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      EQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      EQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
2:      EQ,SQ
28:     LQ
80380:  Q
20:     Q,LQ
35824:  Q,Q
8:      Q,Q,LQ
15651:  Q,Q,Q
5:      Q,Q,Q,LQ
6335:   Q,Q,Q,Q
1:      Q,Q,Q,Q,LQ
2739:   Q,Q,Q,Q,Q
1166:   Q,Q,Q,Q,Q,Q
579:    Q,Q,Q,Q,Q,Q,Q
293:    Q,Q,Q,Q,Q,Q,Q,Q
124:    Q,Q,Q,Q,Q,Q,Q,Q,Q
93:     Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
46:     Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
28:     Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
10:     Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
9:      Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
4:      Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
4:      Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
2:      Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
2:      Q,Q,Q,Q,Q,SQ
1:      Q,Q,Q,Q,SQ,Q,Q,Q,Q,Q
4:      Q,Q,Q,SQ
2:      Q,Q,Q,SQ,Q,Q
1:      Q,Q,Q,SQ,Q,Q,Q
1:      Q,Q,Q,SQ,Q,Q,Q,Q,Q
1:      Q,Q,Q,SQ,Q,Q,Q,Q,Q,Q
17:     Q,Q,SQ
3:      Q,Q,SQ,Q
4:      Q,Q,SQ,Q,Q
3:      Q,Q,SQ,Q,Q,Q
2:      Q,Q,SQ,Q,Q,Q,Q
58:     Q,SQ
12:     Q,SQ,Q
1:      Q,SQ,Q,Q
6:      Q,SQ,Q,Q
9:      Q,SQ,Q,Q,Q
2:      Q,SQ,Q,Q,Q,Q
3:      Q,SQ,Q,Q,Q,Q,Q
2:      Q,SQ,Q,Q,Q,Q,Q,Q
1:      Q,SQ,Q,Q,Q,Q,Q,Q,Q,Q
1:      Q,SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      Q,SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      Q,SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
2:      Q,SQ,SQ
22:     SQ
2:      SQ,EQ
5:      SQ,EQ,Q
5:      SQ,EQ,Q,Q
3:      SQ,EQ,Q,Q,Q
1:      SQ,EQ,Q,Q,Q,Q,Q
58:     SQ,Q
174:    SQ,Q,Q
177:    SQ,Q,Q,Q
102:    SQ,Q,Q,Q,Q
46:     SQ,Q,Q,Q,Q,Q
22:     SQ,Q,Q,Q,Q,Q,Q
9:      SQ,Q,Q,Q,Q,Q,Q,Q
8:      SQ,Q,Q,Q,Q,Q,Q,Q,Q
2:      SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q
2:      SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
2:      SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
1:      SQ,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q,Q
2:      SQ,Q,Q,SQ
1:      SQ,Q,SQ,Q,Q,Q
```

Figure 6.1: The PQP example strings.

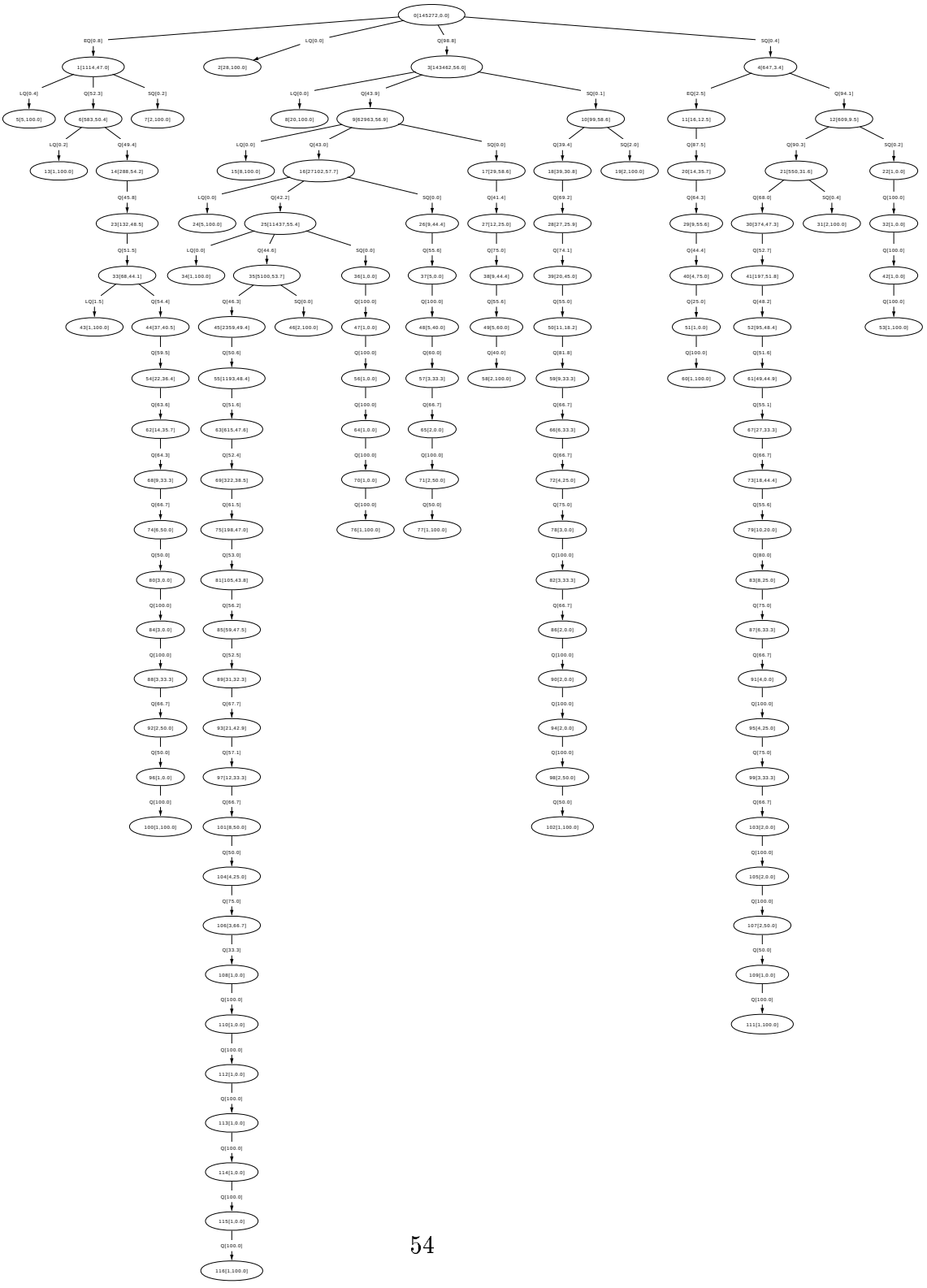

Figure 6.2: The PQP prefix-tree.

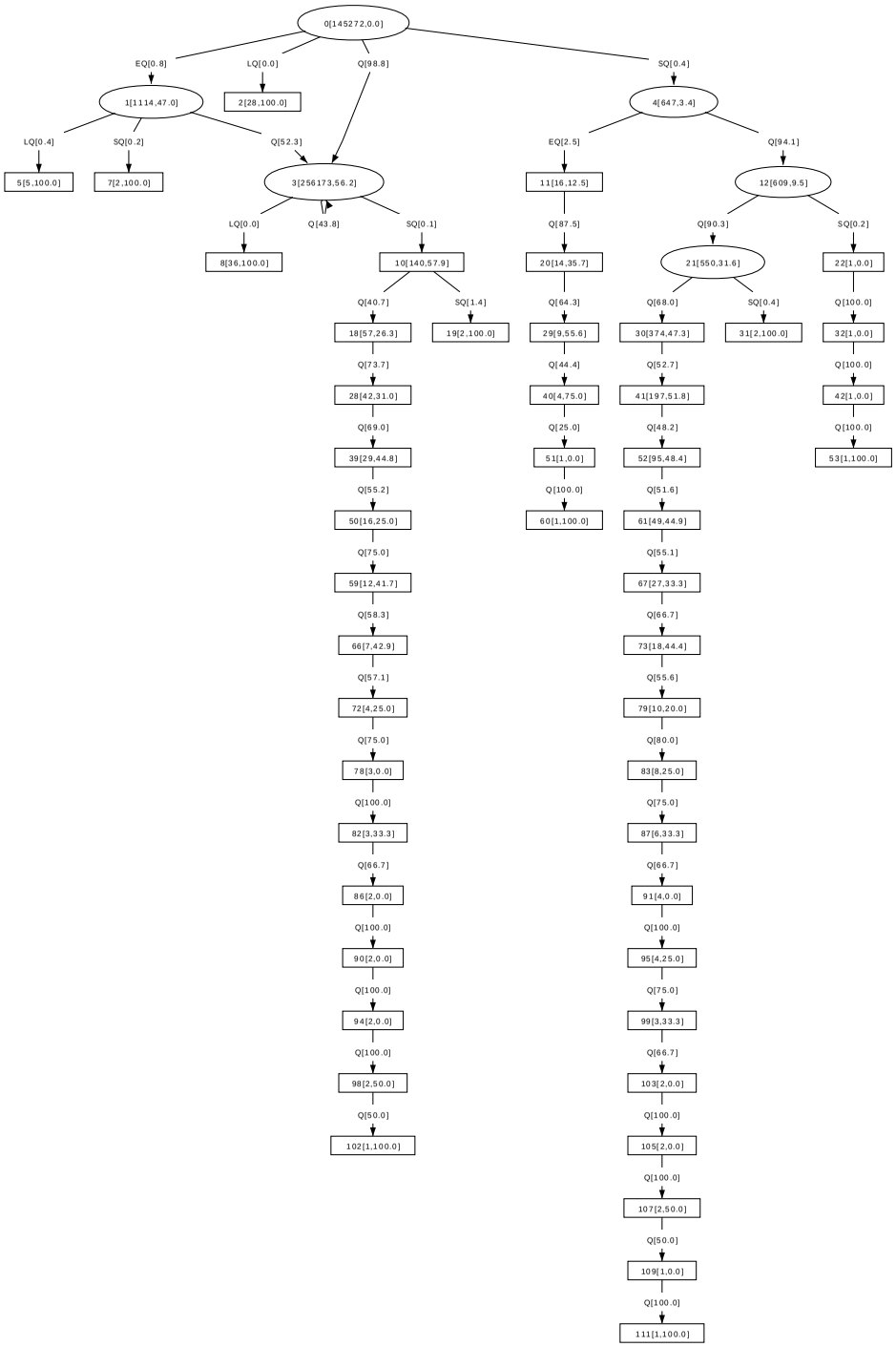

Figure 6.3: The PQP inference result with $\alpha = \beta = \gamma = 0.025$.

Examination of the result reveals two adjustments that can be made to the inference parameters. The first is based on the observation that nodes 1 and 3 are very similar: they both accept an LQ or SQ or any number of Qs, and their transition and termination probabilities all differ by less ten percent. Unless these slight differences are deemed significant enough to represent in the model, it is better to merge the two nodes. This can be done by increasing γ to 0.1, thus allowing nodes to test equal if their true probabilities differ by no more than ten percent. The second adjustment affects nodes 4, 12 and 21. These express the fact that strings starting with an SQ are much more likely to end with more than two Q's. This rule only applies, however, to about five hundred of the over one hundred and forty five thousand PQPs in the dictionary. If we choose to simplify the model at the expense of a small amount of inaccuracy for these cases, we can reduce α and β to reclassify these nodes as low frequency. Trial and error reveals that this can be accomplished with $\alpha = \beta = 0.005$.

The result after application of the two adjustments described above is shown in Figure 6.4. The next step is to do something with the low frequency components. Merging every low frequency tree into the first node that can parse it gives the result in Figure 6.5. Tentative transitions in that diagram are marked with dashed lines.

Based on inspection of the graph, a potential problem can be identified with the transition from node 1 to 0 on input of SQ. That transition creates a cycle that allows strings to contain more than one EQ, a situation that cannot occur in the dictionary. Repointing the tentative transition to the next node that can parse its low frequency subtree gives the automaton in Figure 6.6. Based on previous knowledge of the text, that result has been accepted as a good model for the PQP element.

6.3 Understanding Example

Some graph visualization and string listing techniques were implemented. These included two methods appropriate for general understanding: pruning components of the graph below a frequency threshold and listing strings in probability order.

Two inputs were used. The first was the dictionary entry or E element which has 644 strings and a prefix tree with 1316 nodes. The second was the quotation paragraph or QP element which has 1127 strings and a prefix tree

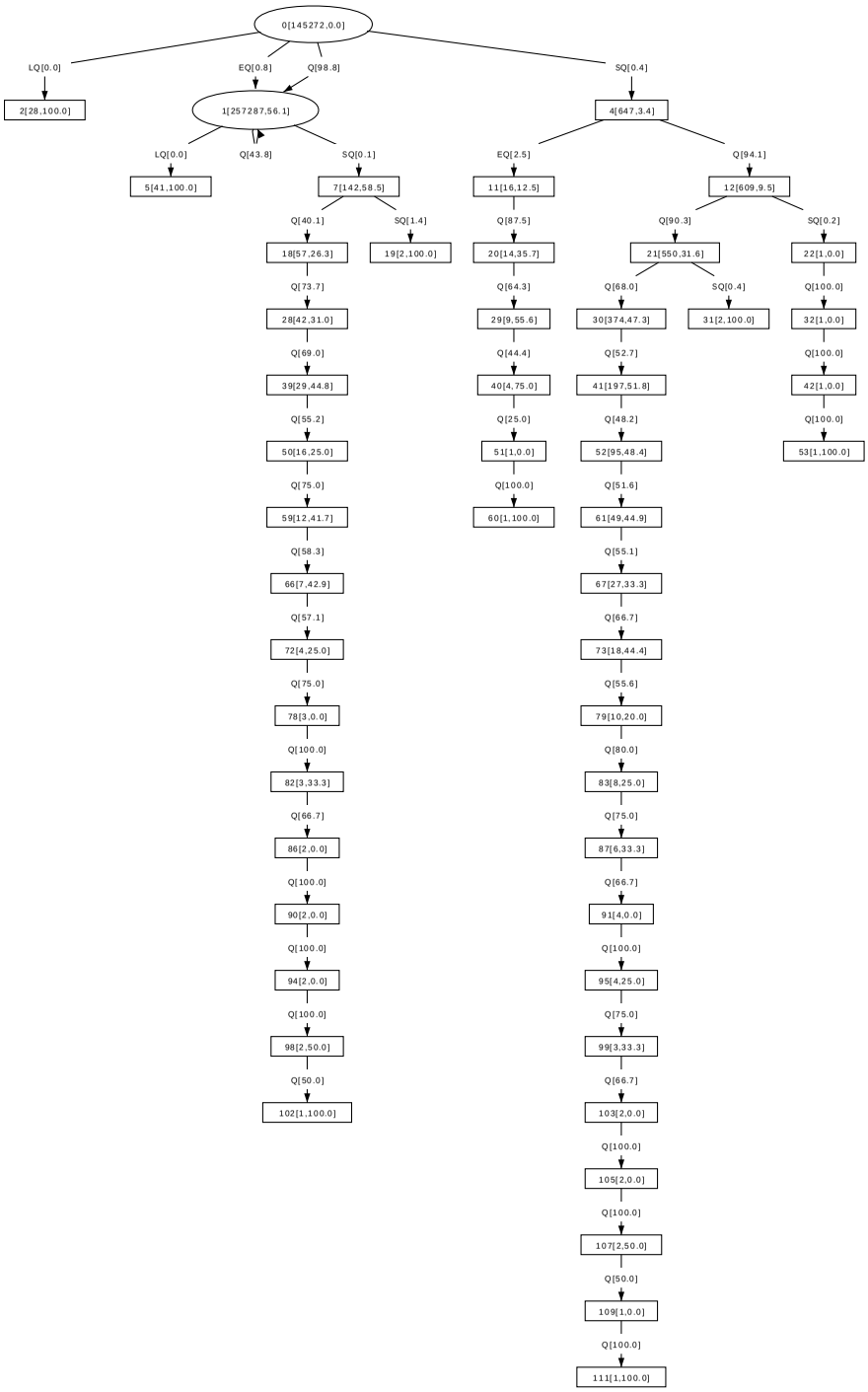

Figure 6.4: The PQP inference result with $\alpha = \beta = 0.005$ and $\gamma = 0.10$.

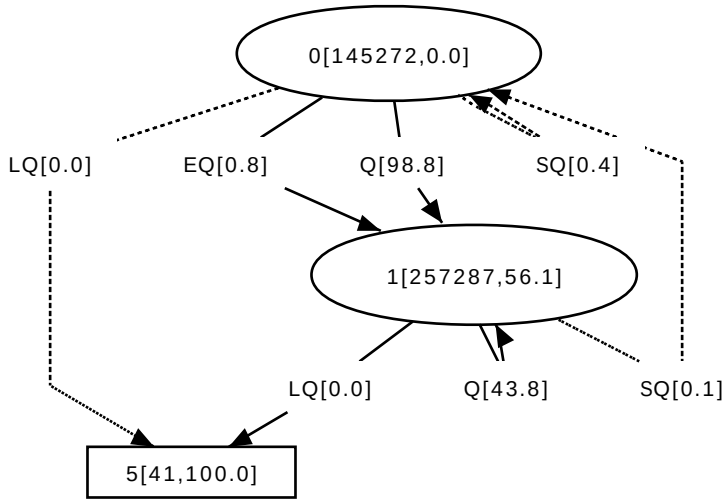

Figure 6.5: Figure 6.4 with low frequency components merged into other parts of the graph.

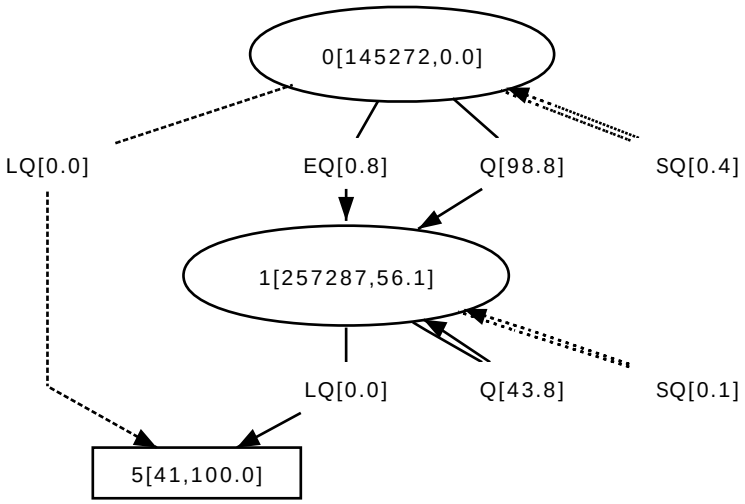

Figure 6.6: Figure 6.5 with the tentative transition from node 1 to node 0 on input of SQ repointed.

with 2601 nodes.

A single application of the algorithm took roughly thirty seconds for both of these inputs. The pruned graphs of the final E and QP results, arrived at after five or six adjustments to the inference parameters, are shown in Figures 6.7 and 6.8. The most frequent strings are shown in Figures 6.9 and 6.10. These high frequency components of the model were judged to be appropriate based on previous understanding of the text. The low frequency components and tentative merges could not be completely evaluated since no appropriate techniques were implemented.

6.4 Applicability of the Representation

It is a concern whether stochastic finite automata reasonably model the semantics of text structure. While representations used for existing grammar specifications are normally no more powerful than regular languages, it is possible to imagine constraints that require the added expressiveness of context free languages. For example, a grammar might be required to express the restriction that documents contain as many citations in the text as entries in the bibliography. In practice, however, restrictions of that type are not usually required.

For stochastic finite automata, the question is whether the implied probability distributions are applicable to text structure. The basic assumption is reasonable: in a given state, possible next elements can be assigned probabilities. It is not obvious, however, whether the probability distributions implied by loops in an automaton are applicable. Consider the simple automaton having a single state that terminates with probability 0.5 and loops back to itself on input of Q with probability 0.5. This generates the language $\{(Q^n, 0.5^{n+1}) \mid n \geq 0\}$ which has a probability distribution geometric in the number of Q's. Is this a useful model, or do repeating symbols in text structure usually follow some arbitrary distribution? If so, then the graph will be more complex: an accurate model of any non-geometric distribution represented as a stochastic finite automaton takes the form of a long string of states rather than a short cycle.

For the *OED* data examined, repeating symbols were found to fit quite well to geometric distributions. That is to say, γ or generalization settings needed to ensure that repeating symbols were represented by cycles did not have to be any higher than necessary to ensure reasonable structures for

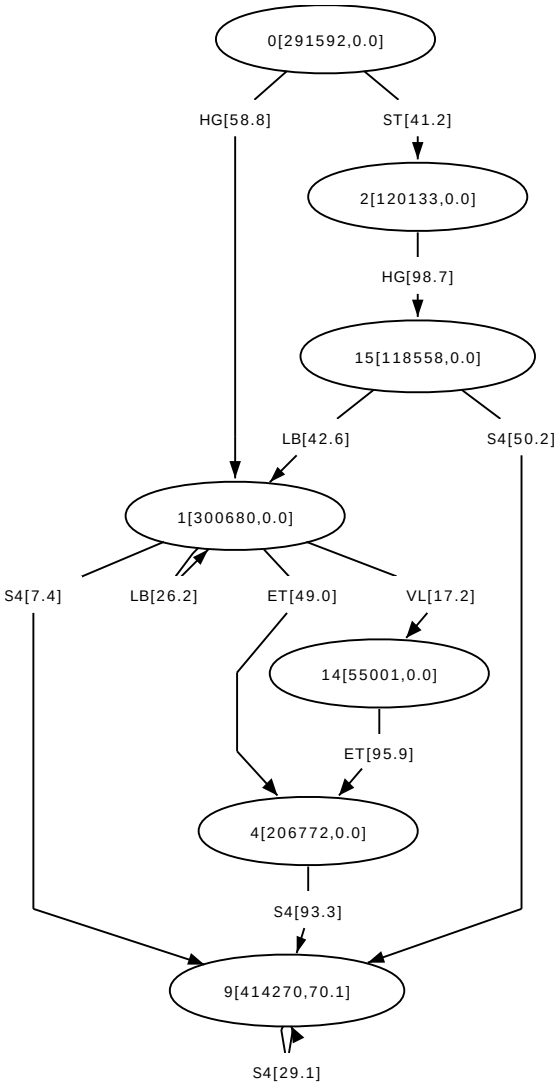

Figure 6.7: The automaton inferred for E. Components below probability 0.05 are pruned. The parameters were $\alpha = \beta = 0.1$, $\gamma = 0.4$.

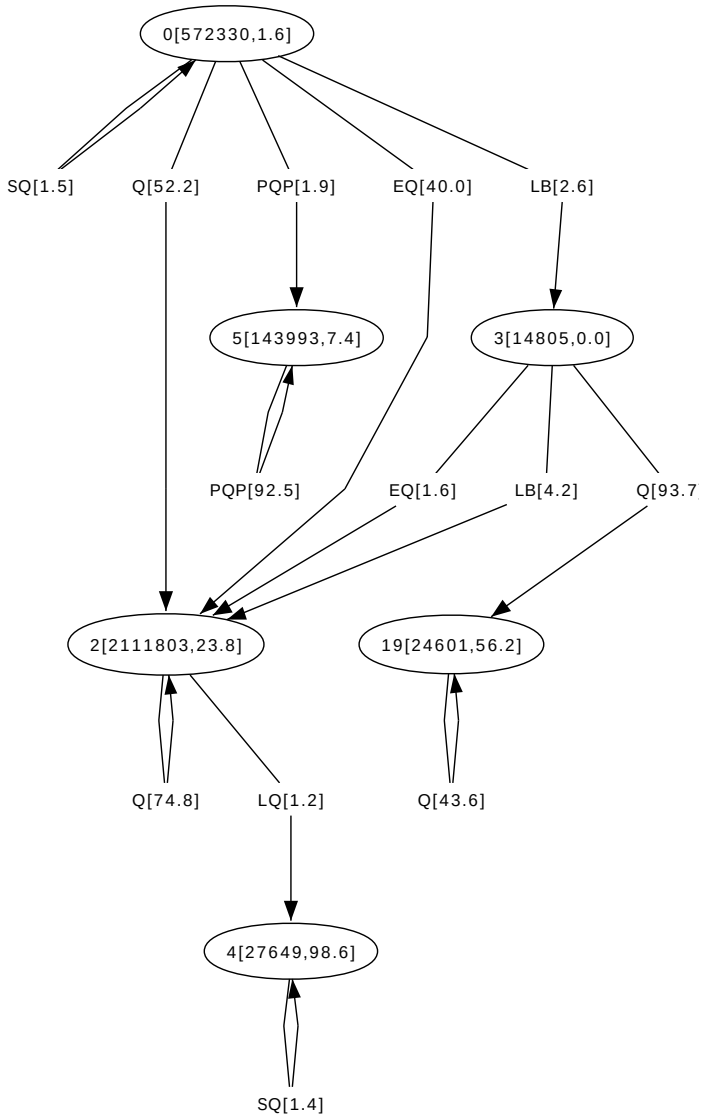

Figure 6.8: The automaton inferred for QP. Components below probability 0.01 are pruned. The parameters were $\alpha = \beta = 0.1$, $\gamma = 0.3$.

```
0.188624 0.154857 HG ET S4
0.143293 0.204186 ST HG S4
0.063461 0.024942 HG VL ET S4
0.055602 0.055578 ST HG LB ET S4
0.054907 0.049216 HG ET S4 S4
0.049357 0.094361 HG LB ET S4
0.041712 0.000093 ST HG S4 S4
0.030566 0.047662 HG S4
0.018707 0.021825 ST HG LB VL ET S4
0.018473 0.013557 HG VL ET S4 S4
0.016606 0.016746 HG LB VL ET S4
0.016185 0.006516 ST HG LB ET S4 S4
0.015983 0.020803 HG ET S4 S4 S4
0.014549 0.048177 ST HG LB LB ET S4
0.014368 0.010933 HG LB ET S4 S4
0.012915 0.009729 HG LB LB ET S4
0.012142 0.000000 ST HG S4 S4 S4
0.011440 0.014308 ST HG ET S4
0.009010 0.007788 ST HG LB S4
0.008898 0.005542 HG S4 S4
0.007998 0.006468 HG LB S4
0.007880 0.009469 HG ET S1 S1
0.007145 0.007853 ST HG VL ET S4
0.005445 0.005436 ST HG LB VL ET S4 S4
0.005377 0.010899 HG VL ET S4 S4 S4
0.005180 0.006650 HG ET S0
0.004895 0.008628 ST HG LB LB VL ET S4
0.004834 0.004314 HG LB VL ET S4 S4
0.004711 0.001403 ST HG LB ET S4 S4 S4
0.004653 0.009726 HG ET S4 S4 S4 S4
0.004345 0.005371 HG LB LB VL ET S4
0.004235 0.001914 ST HG LB LB ET S4 S4
0.004182 0.001715 HG LB ET S4 S4 S4
0.003807 0.001067 ST HG LB LB LB ET S4
0.003760 0.001962 HG LB LB ET S4 S4
0.003536 0.004606 ST ST HG S4
0.003535 0.000000 ST HG S4 S4 S4 S4
0.003380 0.000631 HG LB LB LB ET S4
0.003330 0.002071 ST HG ET S4 S4
0.002726 0.002912 HG VL S4
0.002651 0.005906 HG VL ET S1 S1
0.002623 0.000219 ST HG LB S4 S4
0.002590 0.001636 HG S4 S4 S4
0.002358 0.004523 ST HG LB LB S4
0.002328 0.000429 HG LB S4 S4
0.002323 0.000799 ST HG LB ET S1 S1
0.002294 0.000007 HG ET S1 S1 S4
0.002093 0.000830 HG LB LB S4
0.002080 0.001876 ST HG VL ET S4 S4
0.002062 0.001934 HG LB ET S1 S1
```

Figure 6.9: The 50 most probable strings generated by the inferred model for E. The first number is the probability predicted by the model, the second is the probability in the training set.

```
0.124114 0.088330 Q
0.095091 0.084955 EQ
0.092839 0.090518 Q Q
0.071129 0.056571 EQ Q
0.069444 0.086020 Q Q Q
0.053205 0.055425 EQ Q Q
0.051945 0.069754 Q Q Q Q
0.039798 0.046381 EQ Q Q Q
0.038855 0.051879 Q Q Q Q Q
0.029769 0.034238 EQ Q Q Q Q
0.029064 0.037922 Q Q Q Q Q Q
0.022268 0.023781 EQ Q Q Q Q Q
0.021740 0.026346 Q Q Q Q Q Q Q
0.016656 0.016505 EQ Q Q Q Q Q Q
0.016262 0.018587 Q Q Q Q Q Q Q Q
0.015741 0.015240
0.013623 0.014379 LB Q
0.012459 0.011360 EQ Q Q Q Q Q Q Q
0.012164 0.012853 Q Q Q Q Q Q Q Q Q
0.009319 0.007699 EQ Q Q Q Q Q Q Q Q
0.009099 0.009039 Q Q Q Q Q Q Q Q Q Q
0.006971 0.005409 EQ Q Q Q Q Q Q Q Q Q
0.006806 0.006345 Q Q Q Q Q Q Q Q Q Q Q
0.006355 0.001535 Q LQ
0.005942 0.005864 LB Q Q
0.005214 0.003913 EQ Q Q Q Q Q Q Q Q Q Q
0.005091 0.004604 Q Q Q Q Q Q Q Q Q Q Q Q
0.004869 0.016061 EQ LQ
0.004754 0.001276 Q Q LQ
0.003900 0.002687 EQ Q Q Q Q Q Q Q Q Q Q Q
0.003808 0.003333 Q Q Q Q Q Q Q Q Q Q Q Q Q
0.003642 0.010325 EQ Q LQ
0.003556 0.000804 Q Q Q LQ
0.002918 0.001872 EQ Q Q Q Q Q Q Q Q Q Q Q Q
0.002848 0.002293 Q Q Q Q Q Q Q Q Q Q Q Q Q Q
0.002724 0.005747 EQ Q Q LQ
0.002660 0.000527 Q Q Q Q LQ
0.002592 0.002366 LB Q Q Q
0.002182 0.001406 EQ Q Q Q Q Q Q Q Q Q Q Q Q
0.002131 0.001720 Q Q Q Q Q Q Q Q Q Q Q Q Q Q
0.002038 0.003356 EQ Q Q Q LQ
0.002017 0.002059 LQ
0.001989 0.000314 Q Q Q Q Q LQ
0.001922 0.000587 SQ Q
0.001632 0.000916 EQ Q Q Q Q Q Q Q Q Q Q Q Q Q
0.001594 0.001178 Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q
0.001524 0.001707 EQ Q Q Q Q LQ
0.001488 0.000170 Q Q Q Q Q Q LQ
0.001473 0.000816 SQ EQ
0.001438 0.000980 SQ Q Q
```

Figure 6.10: The 50 most probable strings generated by the inferred model for QP. The first number is the probability predicted by the model, the second is the probability in the training set.

other parts of the data.

Chapter 7

Conclusions

7.1 Summary of Main Results

The main results of this work are in three areas:

1. General improvements to the inference algorithm.

2. Techniques for understanding stochastic models.

3. Insights into the application domain of text structure.

The inference algorithm was improved in two ways. A statistically justifiable test for separating low frequency components of the data was incorporated, and the equivalence test was modified to allow adjustment of the generalization level in an appropriate way. Choices regarding treatment of low frequency components were clearly defined, and one possible interactive approach based on tentative merging was proposed.

Various techniques were considered for understanding, evaluation and interactive feedback. Some graph visualization and string and path listing approaches were implemented and evaluated informally. Several extensions to these approaches were also proposed but not implemented.

Overall, it was found that the semantics of the text structure data examined could be well described by stochastic finite automata. Repeating symbols did conform closely to geometric probability distributions. The modifications to the algorithm addressed two specific observations that were made about the data: 1) many low-frequency components were typically present that needed to be separated from the statistically significant components

and, 2) the data did not behave as if generated by an *exact* stochastic model, but rather, an appropriate inference result had to be determined by varying the level of generalization.

7.2 Future Work

The overall purpose of generating a model is, of course, to use it. Thus, one possibility for future work would be to try using the models generated by this approach for traditional text grammar applications such as validation and editing. Other, novel applications can also be imagined to make use of the stochastic nature of the results. For example, a system could be constructed to assist authors in the creation of documents. This could involve flagging excessively rare structures in the process of their creation or listing possible next elements of partially complete entries in order of their probability.

Several comparisons are probably called for as part of further exploration of the inference approach. The algorithm could be tried with different text, or with different probabilistic modeling applications, to establish whether the *OED* data exhibited typical behavior. It might also be useful to evaluate the predictive power of the generated models by, for example, generating them using half the entries in the *OED* and comparing them to the remainder. Another possibility might be to compare several alternative algorithms to establish whether the stochastic approach in general, and this algorithm in particular, represent any real improvement over existing approaches to text structure inference.

For understanding, some of the proposed techniques need to be implemented, and many others can also be imagined. For example, graph or network analysis techniques for finding strongly connected components or shortest paths could be applied to the results, and might have interesting interpretations in the context of text structure. It might also be useful to convert the automata to stochastic grammar representations, although some way would have to be found to ensure the results were organized for easy understanding.

Other possible improvements could be made in the area of feedback. Currently, parameters have clearly defined interpretations with respect to individual tests, but not to final results. If appropriate relationships could be discovered, it might be better to allow feedback to be given by specifying desired characteristics of the output. For example, the user could require

that particular nodes be merged or classified as low frequency, or that the total number of nodes in the result be less than some specified value.

Bibliography

[AFQ89] J. André, R. Furuta, and V. Quint. *Structured Documents*. The Cambridge Series on Electronic Publishing. Cambridge University Press, 1989.

[AMN94a] Helena Ahonen, Heikki Mannila, and Erja Nikunen. Forming grammars for structured documents: An application of grammatical inference. In Carrasco and Oncina [CO94a], pages 153–167.

[AMN94b] Helena Ahonen, Heikki Mannila, and Erja Nikunen. Generating grammars for SGML tagged texts lacking DTD. In M. Murata and H. Gallaire, editors, *Proc. Workshop on Principles of Document Processing (PODP)*, Darmstadt, 1994. To appear also in Mathematical and Computer Modelling.

[And73] M.R. Anderberg. *Cluster Analysis for Applications*. Academic Press, New York, 1973.

[Ang78] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39:337–350, 1978.

[Ang82] D. Angluin. Inference of reversible languages. *Journal of the ACM*, 29:741–785, 1982.

[Ang92] Dana Angluin. Computational learning theory: Survey and selected bibliography. In *Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 351–369, Victoria, British Columbia, Canada, 4–6 May 1992.

[AS83] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[Ber89] Donna Lee Berg. The research potential of the electronic OED2 database at the University of Waterloo: a guide for scholars. Technical Report OED-89-02, UW Centre for the New Oxford English Dictionary, Waterloo, Ontario, May 1989.

[Ber93] Donna Lee Berg. *A Guide to the Oxford English Dictionary: The essential companion and user's guide.* Oxford University Press, Oxford, 1993.

[Che91] Jinhua Chen. Grammar generation and query processing for text databases, January 1991. Research proposal, University of Waterloo.

[Chr92] Howard B. Christensen. *Introduction to Statistics.* Saunders College Publishing, Fort Worth, 1992. (or any introductory statistics textbook).

[Cla91] David Clark. Finite state transduction tools. Technical Report OED-91-03, UW Centre for the New Oxford English Dictionary and Text Research, Waterloo, Ontario, 1991.

[CM57] N. Chomsky and G.A. Miller. Pattern conception. Technical Report AFCRC-TN-57-57, 1957. (ASTIA Document No. AD 110076).

[CO94a] R. Carrasco and J. Oncina, editors. *Proc. Second Annual Colloquium on Grammatical Inference and Applications (ICGI)*, Lecture Notes in Computer Science 862. Springer-Verlag, 1994.

[CO94b] Rafael C. Carrasco and Jose Oncina. Learning stochastic regular grammars by means of a state merging method. In Carrasco and Oncina [CO94a], pages 139–152.

[CRD87] James H. Coombs, Allen H. Renear, and Steven J. DeRose. Markup systems and the future of scholarly text processing. *Communications of the ACM*, 30(11):933–947, November 1987.

[Fu82] King Sun Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Englewood Cliffs, N.J., 1982.

[FW95] Michael Fröhlich and Mattias Werner. *daVinci 1.4 User Manual*, 1995. available from http://www.informatik.uni-bremen.de/ inform/forschung/daVinci.

[FX93] Peter Fankhauser and Yi Xu. *MarkItUp!* an incremental approach to document structure recognition. *Electronic Publishing – Origin, Dissemination and Design*, 6(4):447–456, December 1993.

[Gol67] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[Gol78] E. M. Gold. Complexity of automaton identification from finite data. *Information and Control*, 37:302–320, 1978.

[Hoe63] W. Hoeffding. Probability inequalities for sums of bounded random variables. *American Statistical Association Journal*, 58:13–30, 1963.

[Hor69] J.J. Horning. A study of grammatical inference. Technical Report CS-139, Computer Science Department, Stanford University, California, 1969.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[ISO86] ISO. Information processing – text and office systems – standard generalized markup language (SGML), 1986.

[Kaz86] Rick Kazman. Structuring the text of the *Oxford English Dictionary* through finite state transduction. Technical Report CS-86-20, University of Waterloo, Computer Science Department, 1986.

[KLMN90] Pekka Kilpeläinen, Greger Lindén, Heikki Mannila, and Erja Nikunen. A structured document database system. In Richard Furuta, editor, *EP90 – Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, The Cambridge Series on Electronic Publishing. Cambridge University Press, 1990.

[MB77] Fred J. Maryanski and Taylor L. Booth. Inference of finite-state probabilistic grammars. *IEEE Transactions on Computers*, C26(6):521–536, June 1977.

[MBCO33] James A.H. Murray, Henry Bradley, William A. Craigie, and Charles T. Onions, editors. *The Oxford English Dictionary*, Oxford, 1933. Clarendon Press.

[Mic90] Laurent Miclet. Grammatical inference. In Horst Bunke and Alberto Sanfeliu, editors, *Syntactic and Structural Pattern Recognition*, pages 237–303, Singapore, 1990. World Scientific.

[Oa92] J. Oncina and P. García. Inferring regular languages in polynomial updated time. In N. Pérez de la Blanca, A. Sanfeliu, and E. Vidal, editors, *Pattern Recognition and Image Analysis*, pages 49–61. World Scientific, 1992.

[SB94] Juan Andrés Sánchez and José Miguel Benedí. Statistical inductive learning of regular formal languages. In Carrasco and Oncina [CO94a], pages 130–138.

[Sha95] Keith Shafer. Creating DTDs via the GB-engine and Fred, 1995. OCLC Fred web page (also in SGML-95?).

[Sil49] G.P. Sillitto. Note on approximations to the power function of the 2x2 comparative trial. *Biometrika*, 36:347–352, 1949.

[SO94] Andreas Stolcke and Stephen Omohundro. Inducing probabilistic grammars by bayesian model merging. In Carrasco and Oncina [CO94a], pages 106–118.

[Val84] L.G. Valiant. A theory of the learnable. *Communciations of the ACM*, 27(11):1134–1142, 1984.

[Vid94] Enrique Vidal. Grammatical inference: An introductory survey. In Carrasco and Oncina [CO94a], pages 1–4.