GRAMMATICAL INFERENCE BY CONSTRUCTIVE METHOD

Dissertation for the Degree of Ph. D.

MICHIGAN STATE UNIVERSITY

JIUNN-I LIOU

1977

This is to certify that the

thesis entitled

GRAMMATICAL INFERENCE BY
CONSTRUCTIVE METHOD

presented by

JIUNN-I LIOU

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in ~~Computer~~ Science

Richard C. Dubes
[illegible]
Major professor

Date 1, 24, 1977

O-7639

G104058

ABSTRACT

GRAMMATICAL INFERENCE BY CONSTRUCTIVE METHOD

By Jiunn-I Liou

In this dissertation a grammatical inference problem is investigated in which constructive methods for inferring finite-state and Chomsky normal form p-grammars from a p-sample are developed. An heuristic approach to grammatical inference is taken. Complexity measures and acceptance criterion are used in selecting a grammar. The solution p-grammar for a p-sample is defined as the p-grammar which has the least complexity and generates an acceptable language.

In the first part of the thesis, a procedure for analyzing sample structure is proposed. Five dissimilarity measures based on the minimal cost of a sequence of edit operations (insertion, deletion, change) are defined and discussed. The type of sequence, determined by the position and the depth of an edit operation in a digraphic representation of paths is used in the assignment of cost. It is shown that these measures vary in the ability to discriminate among strings. A clustering algorithm based on a labelled MST and seven interpretations of inconsistent edges are proposed to break a large inference problem into several small ones. The algorithm uses the notions of "common substrings" and "length" and results in an inference tree. It is argued that the inference tree will provide sufficient information for finding recursive and other string structures, as well as promoting the efficiency of construction.

Techniques used as tools for the development of constructive methods are presented in the second part of the thesis. A size complexity measure is suggested to evaluate the performance of the five dissimilarity measures. A complexity measure and several difference measures based on information theory are defined and used to optimize the construction of p-grammars. The difference measures are shown to be bounded and more realistic than others in the literature. The Kolmogrov-Smirnov tests are suggested to measure the difference between language and sample and are compared with the chi-square goodness of fit test.

Finally, all the above techniques are integrated into constructive methods for the inference of finite-state and Chomsky normal form p-grammars. The method consists of two parts: Construct an initial p-grammar for the p-sample according to the inference tree, then merge productions to generate candidate p-grammars. A set of rules based on six different situations are defined for merging pair-related productions. This constructive method has several advantages over other methods. First, it uses a very general, unique and computationally efficient method for analyzing sample structure. Second, it provides a more realistic difference measure for p-languages than those suggested in the literature. Third, it is reasonable and can be used in very general cases. The method is different from others in the way it analyzes sample structure, creates candidate p-grammars and in its heuristic strategies.

GRAMMATICAL INFERENCE BY CONSTRUCTIVE METHOD

By

Jiunn-I Liou

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Computer Science

January, 1977

ACKNOWLEDGEMENTS

I want to thank each of the members of my doctoral committee for his unique contribution. Thanks go to Dr. Mort A. Rahimi for serving as my academic advisor in the early stage of my program and for continually encouraging me during the course of this research. Thanks go to Dr. Anil K. Jain, Dr. John J. Forsyth, Dr. Lewis Greenberg and Dr. Dennis D. Gilliland for serving on my guidance committee and their critical review of this thesis. My special thanks go to Dr. Herry Hedges, Dr. Carl V. Page and the Division of Engineering Research for providing me with financial assistance during my thesis writing.

Of course the committee member and faculty member I am most indebted to is my thesis advisor, Dr. Richard C. Dubes. Most of my research abilities were developed by him. Without his encouragement, patience and guidance I would never had the chance to complete my degree.

I am also indebted to my wife, Alice Shih-Sing, for her love, patience and understanding throughout my years of study.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued . . .)

TABLE OF CONTENTS (Continued . . .)

TABLE OF CONTENTS (Continued . . .)

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The problem of learning a grammar based on a set of sample strings is called the grammatical inference problem. A survey of literature /27/,/28/ shows that the problem of grammatical inference has recently received a great deal of attention. The importance of studying grammatical inference has been pointed out in several papers /25/,/27/,/28/,/30/, /61/. Based on this substantial information, this thesis will study a grammatical inference problem.

In this chapter, we will introduce some basic notations for grammatical inference, state the grammatical inference problem studied in this thesis, describe the ideas presented, lay out the organization of this thesis and claim its major contributions.

1.1. BASIC NOTATIONS

A number of notations are needed for understanding grammatical inference. These notations are described below.

A phrase structure grammar G, as defined in /32/, is a 4-tuple, $G = (V_n, V_t, R, A)$, where V_n is a finite set of nonterminal symbols; V_t is a finite set of terminal symbols;

A is the start symbol, $A \in V_n$; R is a finite set of production rules of the form:

$B \rightarrow C$, $B \in (V_n \cup V_t)^* - V_t^*$, $C \in (V_n \cup V_t)^*$

The language defined by the grammar G is denoted L(G),

$L(G) = \{x \mid x \in V_t^*, A \overset{*}{\Rightarrow} x\}$

$A \overset{*}{\Rightarrow} x$ means x is derived from A by successive applications of productions in R.

A phrase structure grammar G is context-free if every production $B \rightarrow C$ in R satisfies $\ell(B)=1$, where $\ell(B)$ is the length of the string B.

It is known that /34/ any context-free language can be generated by a grammar in which all productions are of the form:

$B \rightarrow CD$, or $B \rightarrow a$, where $B, C, D \in V_n$, $a \in V_t$.

Grammars of this type are called Chomsky normal form grammars.

A phrase structure grammar G is regular or finite state if every production in R is of the form

$B \rightarrow aC$ or $B \rightarrow a$, $B, C \in V_n$, $a \in V_t$.

A sample of a language L is any subset S of L. The variations "sample S" and "positive sample S^+" will be used synonymously. The sample S is complete if S = L.

A denumerable class of grammars C is said to be admissible if for any $x \in V_t^*$ it is decidable whether or not $x \in L(G)$, for any $G \in C$. A grammar G is compatible with a sample S if $S \subseteq L(G)$. Tree representation is an effective way to represent a class of admissible grammars C, and is defined as: /48/

(1) Each node in the tree corresponds to a grammar G in C.

(2) A node G_t is a descendant of a node G_{t-1} if and only if G_t can be obtained from G_{t-1} under some conditions, such as adding or merging productions.

A probabilistic grammar /27/ (p-grammar) $G_p = (G, P_G)$ is a grammar with production probabilities P_G.

A probabilistic(positive)sample $S_p = (S, P_s)$ of a language L is a sample S with string probabilities P_s.

1.2. GENERAL DISCUSSION OF GRAMMATICAL INFERENCE PROBLEM

The primary problem of grammatical inference can be formulated as follows: Given a sample S infer a grammar G to describe the given sample S and other strings which, in some sense, are of the same nature as S. Precisely the same problem arises in trying to choose a model (theory, function) to explain a collection of sample data.

The problem of inferring finite-state grammars can be approached analytically /3/,/27/,/50/. Since many properties are undecidable about context-free grammars /34/, most studies in inferring context-free grammars are limited to specific types of context-free grammars and often rely on heuristic methods /14/-/16/; /22/,/59/. Two methods, the enumerative method and the constructive method, have been studied in grammatical inference. Enumerative methods simply map the set of positive integers to a class of grammars. Constructive methods are based on the syntactical structure of sample strings. In inferring or guessing a grammar based on a sequence of strings, Gold /30/ has shown that no guessing rules are uniformly better than enumeration. Enumerative methods have been discussed, developed and investigated in the literature /35/,/36/,/46/,/50/,/67/.

To enumerate all possible grammars in a tree, a tree representation for a class of grammars can be defined before enumeration, or an algorithm can be defined which visits all nodes in some order and defines the structure of the grammar found at each node. Enumerative methods admit several variations, such as the formalism of the state-space approach to problem-solving /48/, which involves problem definition, searching strategy and heuristic function. Several enumerative methods have been developed and discussed to infer finite-state and context-free grammars, /50/,/67/, but enumerative methods for p-grammars have not been investigated.

In inferring a grammar from a sample, Fu /27/ has stated that no methods will be more efficient than constructive methods in the sense of constructing an approximate grammar in a reasonable time. Constructive methods have been developed, discussed and investigated in the literature /11/, /16/. The common features of constructive methods are:

(1) Analyze the syntactical structure of the sample strings.

(2) Construct a grammar for the sample to reflect the

syntactical structure.

(3) Merge or add production to get a more acceptable grammar.

In inferring a p-grammar for a p-sample, probability measurements can be used to evaluate the appropriateness of an inferred grammar. Constructive methods also admit several variations, such as methods for analyzing the syntactical structure of sample strings, methods for constructing candidate grammars and methods for testing constructed grammars. Several constructive methods for the inference of finite-state grammars have been proposed /3/,/20/. The inference of context-free grammars by constructive methods for subsets of context-free languages has also been investigated. /14/-/16/, /24/,/61/,/62/. The inference of finite-state /52/ and context-free /11/ p-grammars has been suggested. However, there is no general procedure for analyzing the syntactical structure of sample strings, and no inherent limitations on developing methods for generating candidate grammars. In addition, only a few statistical procedures have been applied to grammatical inference for decision making. Based on these facts, there is a big challenge in developing general procedures for analyzing sample strings, generating candidate grammars, and testing the acceptance of an inferred grammar. This thesis responds to this challenge.

1.3. STATEMENT OF THE PROBLEM AND ASSUMPTIONS

The grammatical inference problem for this thesis is formulated as follows: Given a probabilistic positive sample S_p, construct an acceptable probabilistic grammar G_p to describe S_p. For convenience, we call S_p a probabilistic sample (p-sample).

The basic assumptions for this study are:

(1) The probabilistic sample is finite;

(2) The probabilistic sample is randomly drawn from the source language;

(3) The size of the probabilistic sample is large enough to represent the source language;

(4) The types of grammar being inferred are finite-state and Chomsky normal form p-grammars.

1.4. ORGANIZATION OF THIS THESIS

This thesis includes six chapters. The first chapter introduces the concept of grammatical inference, defines the grammatical inference problem for this thesis and claims the major contributions of the thesis. The second chapter reviews appropriate literature and discusses the most important methods and techniques related to this thesis. The third chapter defines and discusses methods for analyzing the syntactical structure of sample strings. The fourth chapter defines grammatical complexity measures and acceptance criteria for grammatical inference. The fifth chapter discusses the problem of generating candidate grammars and presents the procedure for constructing candidate finite-state and Chomsky normal form p-grammars. Comparisons are then made between this procedure and other methods. The sixth chapter draws conclusions and suggests further research.

1.5. CONTRIBUTIONS OF THIS THESIS

The main contributions of this thesis are listed as below.

(1) The models, methods and literature of grammatical inference are thoroughly reviewed and discussed.

(2) A new, general and reasonable method is proposed for analyzing the syntactical structure of sample strings.

(3) The concept of dissimilarity measure is applied to grammatical inference, so that sample strings can be analyzed in a new dimension.

(4) Five different dissimilarity measures for discriminating between strings are developed, so that strings with similar syntax can be identified.

(5) For the first time, the concept of clustering is applied to grammatical inference, based on a labelled minimum spanning tree (MST). The resultant inference tree provides the framework necessary for computationally efficient grammatical inference.

(6) The role of grammatical complexity in constructive methods for grammatical inference is thoroughly discussed and analyzed.

(7) Several difference measures between p-languages are defined and discussed. Modifications are suggested to make differences between p-languages more realistic.

(8) The application of non-parametric statistics to grammatical inference is expanded, and the Kolmogrov-Smirnov maximum deviation test is introduced and compared to the chi-square test.

(9) A general and practical constructive method, guided by an inference tree, is developed for inferring finite-state and Chomsky normal form p-grammars.

CHAPTER II

LITERATURE REVIEW

2.1. INTRODUCTION

Our purpose in this chapter is to outline the grammatical inference methods already developed by representing their central concepts, results and techniques. We do not attempt to present each method in its entirety, nor to present them in historical order. Rather we concentrate on exhibiting those concepts and techniques which are important to this thesis, and on presenting them from an unified point of view.

The most significant application of grammatical inference methods has been to syntactical pattern recognition /25/. Other application areas include information retrieval /62/, programming language design /16/, translating and compiling /22/, graphic languages /39/, man-machine communication /54/, and artificial intelligence /23/.

Three comprehensive surveys of grammatical inference /27/,/28/,/2/, have laid out the most important work in this area. In this chapter, we will be presenting grammatical inference models, probabilistic concepts, measurement concepts and inference methods for both non-probabilistic and probabilistic grammars, concentrating on those ideas related to the work in this thesis.

2.2. THEORETICAL MODELS

In this section, we will formulate the basic theoretical framework for solving the problem of grammatical inference as it exists in the literature.

The first model of language identification was introduced by Solomonoff /61/, called inductive inference. The model consists of a teacher, a set of sample strings and an inductive procedure. This model only discovers certain recursive productions for a context-free grammar.

The concept of language learnability was first formulated by Gold /30/. He introduced exact language identification which infers an exact, non-probabilistic grammar for an unknown language. The model consists of three components; a definition of learnability, a method of information presentation, and a naming relation which assigns names to languages. A language L is said to be identified in the limit, if a grammatical inference machine, M, exists which guesses a name for L each time it receives a unit of information and for which the guesses are all the same and are a name of L after a finite time.

Gold also defined two other learnabilities: finite identification and fixed time identification. Two basic methods of presenting information are proposed: text presentation and informant presentation.

Gold's main results concern the conditions for information presentation under which a language can be identified in the limit by enumeration. The results show a great difference in the classes of languages that can be identified in the limit by effectively enumerating a class of admissible grammars from two different methods of presenting information.

The exact language identification has several disadvantages such as only being valid through an informant presentation. In addition, inference procedures based on this model are limited to enumeration.

Feldman /21/ introduced a weaker notion of learnability called language approachability, which extends the class of languages learnable from a positive information sequence (text), reduces the information needed to identify a language, and enlarges the domain of language identification. This model requires a complexity measure on a class of grammars, so that the class of grammars can be effectively enumerated in the order of their complexities. The concept of inferring the best grammar for a language with regard to grammatical complexity is practicable because of the time and information needed to identify a language. Feldman's main results show that for any class C of grammars ordered by complexities, there is a machine which can infer the best grammar for any finite positive sample.

Wharton /68/ presented the theory of approximate language identification which is analogous to the existing theory of exact language identification. Approximate language identification is an extension of language approachability.

Defining different metrics on languages, makes the difference between languages measurable, and indicates the degree of approximation between two languages. The difference between languages can be used as an acceptance criterion for an inferred grammar. With the help of grammatical complexity and language difference, grammatical inference problems can be solved by either an enumerative method or a constructive method. Approximate language identification is more practical than other models, not only because various kinds of procedures are applicable, but also because it only requires that the solution grammar generates a language which approximates the unidentified language. In most cases, the best grammar for a language can be found in a reasonable time.

In this research, we use the concept of approximate language identification to construct the best grammar for a set of sample strings, with respect to grammatical complexity and difference between languages. Therefore, the approximate language identification model will be the most significant model in this research.

2.3. PROBABILITY CONCEPTS

Statistical information is as important as structural information, when inferring a probabilistic grammar based on a finite sample. For the past few years, probabilistic grammars and probabilistic automata have received increasing attention /9 /,/26/,/64/,/65/. The mathematical formulations of probabilistic grammars and probabilistic automata have been studied in several papers /19/,/57/,/58/,/63/. The relations between probabilistic automata and probabilistic language have been described /29/. The generative capacity of grammars can be measured by Shannon's entropy /59/. The information-theoretic concept of entropy of a context free language and its relation to the structure generating function have been investigated /43/. The information-theoretic concept of entropy has been related to probabilistic grammars /63/. Soule /63/ has calculated the entropies of a derivation of a sentence and the average terminal symbol and has proposed methods to maximize the information rate.

Booth and Thompson /9/, presented two methods for assigning probabilities to words in a language, developed several properties of probabilistic languages, and investigated the conditions under which a p-grammar is consistent. Horning /36/ and Patel /52/ presented maximum likelihood estimation techniques to determine production probabilities from an experimental set. The concept of strong-approximation of a probabilistic language was introduced by Booth /7/. Other type of approximations based on some type of distance measure were developed by Maryanski /46/. Horning /36/, has described an acceptance technique based on the concept of hypothesis testing. In his method, a finite set of candidate probabilistic grammars is given and one must be selected from the set of grammars which best describes the experimental set.

In this section, we will relate these works to this thesis.

2.3.1. MAXIMUM LIKELIHOOD ESTIMATION TECHNIQUES

In inferring a p-grammar from a p-sample, it is necessary to minimize the difference between the resultant p-language and the p-sample. The p-language should match the p-sample, both in probability and structure. Minimizing this difference is the same as estimating the production probabilities for a grammar from the p-sample. Maximum likelihood estimates of the production probabilities for a p-grammar G maximize the probability of a string being generated by G. This method is a reasonable method for estimating the production probabilities of unambiguous grammars. This is not true for ambiguous grammars, since there exists more than one leftmost derivation for some strings in the language. This simple method can be described as follows:

Let $G = (V_n, V_t, R, P, A)$ be an unambiguous p-grammar, and $E = \{ (x_1,C_1),\ldots\ldots,(x_n,C_n) \}$ be the sample information where $x_i \in S$ and C_i is an associated estimated or a-priori probability of x_i. Let $V_n = \{ A,A_1,\ldots,A_r \}$ be the finite set of nonterminals, and P_{ij} be the probability of the production $A_i \rightarrow B_j$, $B_j \in (V_n \cup V_t)^+$ of G. Let P_{ij} be the estimated production probability for P_{ij}. Under single class estimation, the maximum likelihood estimate for P_{ij} is

$$P_{ij} = n_{ij}/\sum_j n_{ij}, \text{ where } n_{ij} = \sum_{x_k \in S} C_k N_{ij}(x_k)$$

and $N_{ij}(x_k)$ is the number of times that the production $A_i \rightarrow B_j$ is used in generating x_k.

For multiclass estimation, the procedure is similar and will not be presented here.

2.3.2. ACCEPTANCE TESTING OF CANDIDATE GRAMMARS

The purpose of statistical tests in grammatical inference is to choose a p-grammar that can adequately describe the given p-sample. The only two statistical tests that have been used in grammatical inference are Bayesian acceptance /51/ and the chi-square test /12/. There is a functional difference between these two tests. The chi-square test is used to judge the degree of closeness between a p-language and a p-sample, while the Bayes test is used to select the best grammar from a class of grammar for the sample. The Bayes' test requires a prior probability distribution over a class of grammars which is not compatible with the procedure presented for this thesis. The chi-square test is used to test the difference between the theoretical and observed distributions of a sample /46/ and is described below.

Let e_i be the expected number of times that $x_i \in S$ appears in the sample S, and let n be the size of S, and let k be the number of distinct categories. Thus $e_i = n\Pr(x_i)$. The d^2 statistic measures the difference between the expected frequency (e_i) and the observed frequency (c_i) and is defined as:

$$d^2 = \sum_{i=1}^{k} (c_i - e_i)^2/e_i$$

Under general conditions, the distribution of d^2 can be closely approximated by a chi-square distribution with k-1 degrees of freedom (df) /12/.

An hypothesis testing problem can be stated as:

H_0: $e_i = c_i$ for all i, $1 \leq i \leq k$

H_1: $e_i \neq c_i$ for some i, $1 \leq i \leq k$

Let h^2 be the acceptance criterion for a chi-square test. Then the decision rule under a 0-1 loss function is:

Accept H_0 if $d^2 < h^2$, or Reject H_0 if $d^2 \geq h^2$

A meaningful application of the chi-square test requires that the frequency of every string in S should be no less than 5 or 10. In inferring a p-grammar for a p-sample, it is possible that some words may occur with low frequencies. In such a case, the chi-square test may not find an appropriate grammar for the sample. The temptation may arise to make the sample size extraordinary large. However, the chi-square test becomes very sensitive to small differences as the sample size becomes very large. The properties of the chi-square test and the disadvantages when used in grammatical inference will be discussed in Sec. 4.5.2.

2.3.3. DIFFERENCE MEASURE FOR LANGUAGES

The difference between languages is used to evaluate the appropriateness of an inferred grammar for a set of sample strings. There are at least three different approaches to the notion of difference measure of languages.

The first approach to measuring differences between languages is given by Cook /11/ and is called the <u>discrepancy measure</u>. It is an information theoretic measure and based on the probability distributions of the sample and the language (Cf. Sec. 4.4.1.). A second type of difference measure was proposed by Wharton /68/, who defined metrics on the class of all languages $U = \{ L | L \subset V_t^+ \}$ over a finite terminal vocabulary V_t. He considered two types of metrics, the <u>discrete metric</u> and the <u>weight metric</u>. The weight metric is intuitively reasonable in grammatical inference by constructive methods, since the grammar inferred is an approximation to an exact grammar for the sample. Assigning weights to strings in a language is difficult if the language is probabilistic.

The third type of difference measure is based on the string probability /46/. This difference measure only considers the difference of string probabilities over all strings in one language which will reduce the accuracy of the difference.

Each of the three different types of difference measures has some disadvantages. Other difference measures are possible, but it is difficult to establish another dimension that can be used to measure the difference between languages. The difference measure between p-languages based on information theory will be most significant to this thesis, since it involves both syntactical and probability differences.

2.4. GRAMMATICAL COMPLEXITY MEASURE

The idea of grammatical complexity can be used to select a good grammar to represent a sample /21/,/67/. Grammatical complexity measures have received increasing attention over the past few years /5/,/6/,/13/,/32/,/56/. Two different categories of grammatical complexity measures are those based on the size of grammars, and those based on information theory. In this section, we will present the concepts underlying this idea.

2.4.1. COMPLEXITY MEASURE BASED ON SIZE

The size measure of machines is based on the number of states in the machine /5/. The relative size of two machines is independent of the particular measure used /6/. The measure of complexity is the number of steps taken by machines to compute a function.

Wharton /67/ uses the concept of size measure in approximation language identification (Cf. Sec. 4.3.1.). He introduces the maximum length of the right hand sides of productions as a parameter in the complexity measure. Thus the class of context-free grammars may be distinguished by their special normal form of productions. Feldman /21/ defined a general grammatical complexity measure (Cf. Sec. 4.3.2.). Based on this measure, there are many ways to define grammatical complexity measures, but there is no optimal grammatical complexity measure among all complexity measures.

Feldman's and Wharton's complexity measures are for the purpose of effectively enumerating a class of grammars. They may also be used for other purposes such as evaluation. Salomma /56/ defines the index of a grammar and the index of a language which is similar to size measure. Gruska /32/ presented several classifications of context-free languages which are analogous to the size measure.

2.4.2. COMPLEXITY MEASURES BASED ON INFORMATION THEORY

The information conveyed by an event is the decrease in uncertainty that comes about when one of the events does occur. The average information conveyed by a complete set of events is the entropy. Shannon /59/ notes several properties required of a measure of the information conveyed by a complete set of events. Cook /11/ defined information-theoretical complexity and discrepancy measures for grammatical inference.(Cf. Sec. 4.3.4.) Cook's complexity measure assumes that the complexity of a context-free p-grammar is the sum of production complexities. Since context-free grammars have a single nonterminal on the left hand side of each production, the production complexity is determined by the right hand side. This assumption is reasonable only if productions in context-free grammars are statistical independent. The complexity measure only considers the complexities contributed by two sources: the production probability and the right hand side of each production. That is not enough, since there exists some degree of similarity among productions of any context-free grammar. Two context-free grammars can have productions very similar to each other in length and symbol distribution. However, all productions in one might be of the same type, as in a pivot grammar, while those in the other can be heterogeneous. Certainly, this fact should influence complexity.

Horning /36/ suggested that if complexity is defined as a monotonic function of the negative logarithm of the conditional probability $\Pr(G_i|S,C)$, where $G_i \in C$, and C is a class of p-grammars which generate the sample S, then the intrinsic complexity of the grammar and the relative complexity of the sample given the grammar are defined as:

$$- \log \Pr(G_i) \text{ and } - [\log \Pr(S|G_i) - \log \Pr(S)]$$

This is analogous to Cook's complexity measure, since it considers both the information needed to select the grammar G_i from C, and the information needed to generate the sample S. This complexity measure is different from Cook's complexity measure. The complexity of a grammar $G_i \in C$ depends on $\Pr(G_i)$ and $\Pr(S|G_i)$, while Cook's complexity measure considers productions of a p-grammar.

Feldman, et al. /24/ define a derivational complexity of a set of strings relative to a grammar. A real valued function similar to a probability function is defined over the productions of a grammar. The derivational complexity of a grammar is based on the values assigned by the function to productions. This complexity measure is similar to Horning's complexity measure, since they all depend on the probability generated by a p-grammar.

Three different complexity measures based on information theory can be thus used to measure the complexity of a p-grammar. Horning's complexity measure requires the prior probability $Pr(G_i)$ and the conditional probability distribution $Pr(S|G_i)$. Feldman's complexity measure only concerns the derivational complexity of a sample relative to a grammar. Cook's complexity measure ignores the similarity among productions in a grammar. Therefore a complexity measure should be defined which is similar to Cook's complexity measure but will consider the difference among productions. Also, other complexity measures based on information theory can be defined that will possess desirable properties.

An information measure is completely different from a size measure. Different complexity measures can be defined for different purposes and interests. For measuring the complexity of a p-grammar, an information measure is better than a size measure since it considers the production probabilities as well as the productions themselves.

2.5. CONSTRUCTIVE METHODS FOR GRAMMATICAL INFERENCE

The various grammatical inference methods developed in the literatures can be classified into two categories: enumerative methods and constructive methods. Since this thesis studies constructive methods, enumerative method /33/

/35/,/36/,/46/,/48/,/67/, will not be discussed. Constructive methods are based on the structural and statistical properties of the strings in the sample. In this section, we will discuss some of the most important work in this area.

A constructive method for inferring a recursive, unambiguous finite-state grammar from a sample S has been presented /20/. The procedure first constructs a grammar that generates exactly S, then produces a simpler, recursive grammar, that adds more strings to the generated language. Two constructive methods based on a sample S has been described for finite state grammars /3/. The algorithms will infer a grammar which generates S and strings similar to S. The first algorithm /3/ creates and compares sublanguages $S_w = \{x | wx \in S\}$. All strings in S are grouped together when they have the same initial substring, w. Let S_w be the one sublanguage, say the ith. A production $A_i \rightarrow aA_j$ is created, if S_{wa} is the other sublanguage, say the jth, and $A_i \rightarrow a$ is created if $wa \in S$. The second algorithm /3/ is called a linear grammatical inference program, and operates analogously to the finite-state program.

A constructive method for the inference of finite-state p-grammar has been presented /52/ which starts with a small subset H of the positive sample S and generates the set of all possible deterministic derived grammars that could describe this subset. Then it uses the maximum-likelihood method to assign production probabilities. The search is terminated if one of the grammars is accepted according to a chi-square test. Otherwise, the size of H is enlarged. The search continues until either an acceptable grammar is found or the subset H is equal to S.

A method for discovering the nesting or recursive structure in a context-free grammar has been described /61/, /62/. The procedure cannot find the recursive property of a context-free grammar with self-embedding, e.g. $A \rightarrow aAb$, $a \in V_t$, $b \in V_t$, $A \in V_n$. A constructive method for inferring a pivot grammar has been described /24/. The strategy is to find self-embedding in the positive sample. The inference of operator precedence grammars from a structural information sequence of a language has also been presented /14/-/16/.

One of the most important constructive methods for the inference of context-free p-grammars is the hill-climbing method /11/. In this procedure, a cost function is defined that measures the complexity of a grammar and the discrepancy between languages. The procedure starts with the grammar in which the initial symbol goes to each string in the sample with the given string probability. It then examines possible simplifications of the grammar interatively. The inference procedure makes the simplification that yields the grammar of the lowest complexity, provided that it lowers the cost. The process continues until no simplifications that lead to lower-cost grammars are possible.

Another semi-heuristic constructive method for the inference of context-free p-grammars has been developed /46/

in which candidate grammars are generated by searching for various structural features of the sample, such as the "uvwxy" property /34/ of context-free grammars, and constructing corresponding grammars. This procedure is satisfactory for simple grammars. If the original grammar which generates the sample is ambiguous, the language generated by the inferred grammars are statistically close to the sample, but in some cases their productions are quite different.

The problem of grammatical inference by constructive methods is still open for the following reasons:

(1) Other heuristic strategies exist for constructing grammars.

(2) No best evaluation procedure for improving the selection of grammars exists.

(3) No best difference measure between languages for deciding the appropriateness of a language exists.

2.6. SUMMARY

In this chapter, we have summarized the most important work in the area of grammatical inference. The concepts of exact language identification and approximate language identification have been described. Two major approaches to grammatical inference have been presented and recent work on the inference of probabilistic and non-probabilistic grammars have been summarized.

Grammatical inference is still in its infancy. Much theoretical and experimental research needs to be done before we can establish the principles underlying grammatical inference. Many problems are still open especially in the inference of probabilistic and high dimensional grammars. A big challenge exists in grammatical inference by constructive methods, and the inference of context-free grammars.

CHAPTER III

ANALYSIS OF SAMPLE STRINGS

3.1. INTRODUCTION

The purpose of this chapter is to present efficient and general procedures for analyzing the syntactical structure of a given set of sample strings, called the sample structure. The results obtained from these procedures will provide a framework for constructing a grammar whose language is similar to the given set of sample strings. (Cf. Chap. V)

In the literature, a grammar for a set of sample strings is inferred in two steps. First, construct a grammar for each individual string. Then, merge all grammars into a single grammar. The number of grammars and productions constructed during the first step increases linearly with the size of the sample. Feldman /2/ proposed four steps for inferring grammars for a set of sample strings. His first step is to analyze the syntactical structure of the given set of sample strings. This can be accomplished in many ways and will be discussed in Sec. 3.2.

In this chapter, we will propose a cluster analysis method which partitions the set of sample strings according to their similarities and creates an inference tree. Then we will construct a grammar for each cluster. This procedure substantially reduces the number of grammars and productions constructed in the first step. Also, the cluster analysis will suggest recursive structures for strings which will indicate recursive productions for the inferred grammar.

The cluster analysis is based on a measure of proximity between strings. A dissimilarity (distance) measure for strings of different lengths will be proposed in Sec. 3.3. The dissimilarity measure is defined as the minimum cost sequence of "edit operations" needed to change one string to another /49/,/66/. The inadequacy of assigning uniform cost for edit operations regardless of the position of the operation in the operation sequence will be discussed. Alternative ways to assign cost for edit operations will be presented, and their properties will be studied. For the purpose of detecting certain syntactical relations between strings, the dissimilarity measure will be modified by letting the weights for edit operations depend on their positions in the operation sequence.

A cluster analysis method based on the minimal spanning tree is a powerful and general tool for detecting and describing the structure of point clusters /69/. In Sec. 3.4. we will describe this clustering procedure and discuss the significance of cutting inconsistent edges in the minimal spanning tree so that a number of meaningful clusters may be formed.

3.2. SYNTACTICAL STRUCTURE OF SAMPLE STRINGS

The syntactical structure of a finite-state language is different from that of a context-free language. In this section, we will review the methods which have already been developed to analyze the syntactical structure of a set of sample strings for inferring both finite-state and context-free grammars.

The analysis of the syntactical structure of a finite-state language is based on the particular form of productions for finite-state grammars (see Sec. 1.1.). Two methods have been proposed to analyze the syntactical structure of a finite-state language. One uses the idea of formal derivative /10/. The other uses the idea of K-tails of a derivative /27/. These methods are described as follows:

The formal derivative of a set of strings S with respect to the symbol $a \in V_t$ is defined as:

$$D_a S = \{ x | ax \in S \}.$$

As an example, let $S = \{ 01, 100, 111, 0100 \}$. Then

$$D_0 S = \{ 1,100 \},\ D_1 S = \{ 00,11 \}.$$

Based on the formal derivative, a canonical derivative finite-state grammar can be defined for a sample S /52/. The canonical derivative grammar can be used to define a class C of derived grammars /52/. In many situations, the set C will not contain the grammar that generated the sample S /27/.

Let $u = a_1 a_2 \ldots a_n \in V_t^*$, and let $S \in L(G)$. The k-tails of S with respect to u is defined as:

$$g(u,S,k) = \{ x \mid x \in D_u S, \ |x| \leq k \}.$$

For example let $S = \{ 01, 100, 111, 0100 \}$. Then

$g(0,S,2) = \{1\}$, $g(0,S,3) = \{ 1,100\}$, and

$g(1,S,2) = \{ 00,11 \}$.

The k-tails method can be used to define equivalence classes on the states of the canonical derivative finite-state grammar generated by a complete sample S /27/. This method can only obtain rough solution grammars for the sample S.

The k-tails of a finite-state language is based on the formal derivative. These two methods are similar and lead to similar results. These methods can be easily programmed and used to analyze a set of sample strings from a finite-state language. But they are limited to sequentially scanning characters in strings from a sample. An adequate syntactic structure for the sample cannot be discovered by these methods. In addition, grammatical inference methods based on these methods are constricted in their procedures for merging productions. We want to develop a general technique for analyzing syntactic relations among strings so that a more reasonable sample structure can be found and mergers of productions are clear, flexible and permit heuristic strategies.

The syntactical structure of a context-free language is very complicated. To date, only limited research has been done in investigating subsets of a context-free language, such as operator precedence grammars and pivot grammars. For inferring operator precedence grammars, each string in the sample is evaluated to establish a set of precedence relationships that exist on the elements of the strings /14/-/16/. For inferring pivot grammars, the main strategy is to find self-embedding in the sample strings by using substring relationships /24/. There is no general method for analyzing the syntax of a context-free language. Existing techniques for the analysis of syntactical structure of context-free languages are applicable only to small subsets of the language. Thus we are motivated to develop a general method for analyzing the syntactical structure of context-free languages.

3.3. DISSIMILARITY MEASURE

Using cluster analysis methods to analyze the syntactical structure of a set of sample strings requires a proximity measure which determines the syntactical difference between pairs of strings. Recent research /49/,/66/ leads to a reasonable concept of dissimilarity between strings. The dissimilarity between two strings is defined as the minimum cost sequence of "edit operations" needed to change one string to the other. Under this measure, three edit operations are considered: (1) change one character to another, (2) delete a character from a string, (3) insert a character into a string. A cost is assigned to each edit operation.

In this section we will adopt the dissimilarity measure proposed by Okuda et al. /49/ called "Weighted Levenshtein Distance" (WLD) which defines a distance between words of different length. The WLD between strings is explained in terms of a digraph in Sec. 3.3.1. Wagner /66/ proposed an edit distance between strings which has the same configuration as the WLD, but Wagner used a trace idea to interpret the edit distance. Since the edit distance and the WLD are computed with the same algorithm, we will use the notation of Okuda et al.

3.3.1. WEIGHTED LEVENSHTEIN DISTANCE AND UNIFORM EDGE WEIGHTING

Before defining the dissimilarity measure, we need the following preliminary definitions:

Definition: An edit operation is a pair $(a,b) \neq (\Lambda,\Lambda)$ of strings of length less than or equal to 1 and is usually written $a \rightarrow b$.

Let A, B be strings. $A \Rightarrow B$ means that an edit operation is applied to string A to produce string B. In general, if $A := \sigma a \tau$, $B := \sigma b \tau$ for some strings σ and τ, then $A \Rightarrow B$ if $a \rightarrow b$.

Definition: An edit operation $a \rightarrow b$ is

(1) a change operation if $a \neq \Lambda$, $b \neq \Lambda$;

(2) a delete operation if $a \neq \Lambda$, $b = \Lambda$;

(3) an insert operation if $a = \Lambda$, $b \neq \Lambda$.

The three operations are interpreted consistently as follows:

(1) An insertion operation inserts a character into the right end of a string.

(2) A deletion operation deletes a character from the left end of a string.

(3) A change operation changes the character on the left end of a string and moves the changed character to the right end of the string.

Definition: Let h be an arbitrary cost function which assigns to each edit operation $a \rightarrow b$ a non-negative real number $h(a,b)$. When h has the properties shown below, it is called the uniform cost function.

Let $h(a,b) = p$, $h(a,\Lambda) = r$, $h(\Lambda,b) = q$, $a \neq \Lambda$, $b \neq \Lambda$. The basic dissimilarity measure is defined as follows:

Definition: Let A, B be strings. Then the dissimilarity between A and B is defined as:

$$D(A,B) = \min_{i} (pk_i + qm_i + rn_i)$$

where k_i, m_i, n_i are the numbers of changes, insertions and deletions respectively, needed to transfer the string A to the string B in the i^{th} sequence of edit operations.

As a reasonable cost assignment, we will assume, unless otherwise noted, that $q = r$, $p = q+r$, and the cost of changing a character to itself is zero.

If the lengths of strings A and B are n and m, respectively, then all possible ways that string A can be transformed into string B can be represented on a digraph with (n+1) x (m+1) nodes $s_{i,j}$ and 3mn+m+n edges, called the AB digraph. The digraph can be understood as a rectangle of n rows and m columns and n.m city blocks with a diagonal for each block. In the digraph, (n+m-2) nodes have in-degree one and out-degree three, (n+m-2) nodes have in-degree one and out-degree one, one node is a source, representing string A, one node is a sink, representing string B, and the remaining nodes have in-degree three and out-degree three (see Fig.1).

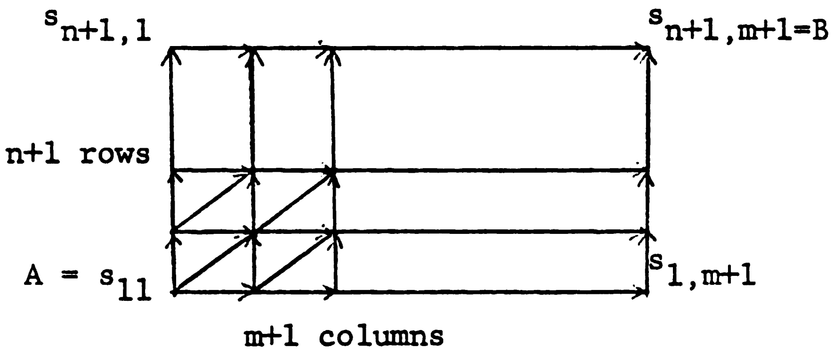


FIG. 1. AB digraph

Each node $s_{i,j}$ in the digraph represents a string, and each edge represents an edit operation. As shown in Fig.1, s_{11} represents the string A, $s_{n+1,m+1}$ represents the string B, and $s_{i,j}$ represents the string composed of the ith to the nth characters of A concatenated with the 1st to the $(j-1)^{th}$ character of B. To obtain $s_{i+1,j}$ from $s_{i,j}$, delete the left end character of $s_{i,j}$ (which is the ith character of A); to obtain $s_{i,j+1}$ from $s_{i,j}$, insert the jth character of B at the right end of $s_{i,j}$; to obtain $s_{i+1,j+1}$ from $s_{i,j}$, move the left end character of $s_{i,j}$ to the right end of $s_{i,j}$ if the left end character of $s_{i,j}$ (which is the ith character of A) is identical to the jth character of B; otherwise change the left end character of $s_{i,j}$ to the jth character of B and moves it to the right end of $s_{i,j}$. These operations are pictured in Fig. 2.

The costs of edit operations are represented as edge weights, as in Fig. 2. Under the uniform cost function h, the edge weights are assigned as follows:

(1) Assign weight r to all edges $(s_{i,j}, s_{i+1,j})$ for $1 \leq i \leq n$, $1 \leq j \leq m+1$, (deletion).

(2) Assign weight q to all edges $(s_{i,j}, s_{i,j+1})$ for $1 \leq i \leq n+1$, $1 \leq j \leq m$, (insertion).

(3) For all edges $(s_{i,j}, s_{i+1,j+1})$, $1 \leq i \leq n$, $1 \leq j \leq m$, if the ith character of A is equal to the jth character of B then assign weight zero; otherwise assign weight p, (change).

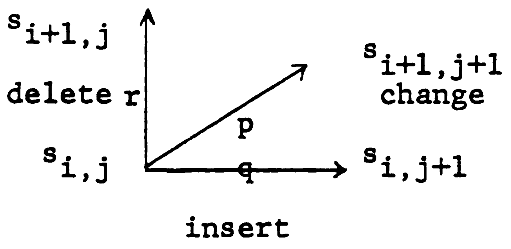


FIG. 2. Uniform Edge Weight

<u>Example 1</u>: Let A : = cb, B : = cab. Then the AB digraph, under a uniform edge weight, is given in Fig.3. For clarity, arrowheads are omitted.

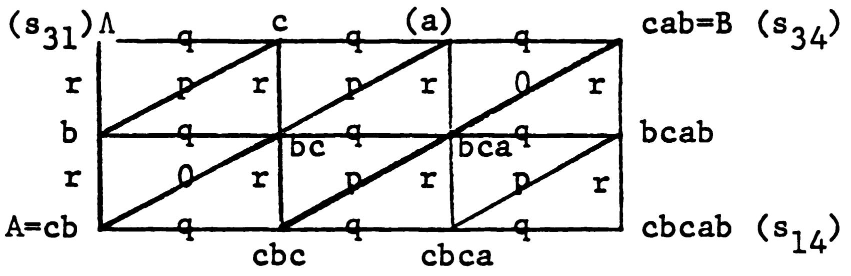


FIG.3. AB digraph under uniform edge weighting

A "path" will mean a directed path from source to sink. The weight of a path in the digraph is the sum of the weights of its edges, and the WLD is the minimum path weight from source to sink.

<u>Example 2</u>: The dissimilarity between A : = cb and B : = cab is $D(A,B) = \min_i (pk_i + qm_i + rn_i)$. Let q = r, p = 2q. Then a minimum weight path is $s_{11}s_{22}s_{23}s_{34}$ with weight q. (See Fig. 3)

Let A,B and C be strings. Then, under uniform cost, the dissimilarity measure has the following properties /51/:

(1) $D(A,B) = 0$ if and only if $A = B$,

(2) $D(A,B) \leq D(A,C) + D(C,B)$,

(3) if $q = r$, then $D(A,B) = D(B,A)$.

The minimum weight path in the digraph can be found with algorithms based on the following two theorems /51/.

Let A, B be strings with length n, m, respectively; p, q, r are the cost of change, insertion and deletion, respectively. Furthermore, let $D_{i,j}$ denote the minimum weight from s_{11} to $s_{i,j}$. Then $D(A,B) = D_{n+1,m+1}$ and it can be computed iteratively.

<u>Theorem 1</u>: $D_{i,1} = (i-1)r$, for $1 \leq i \leq n+1$
$D_{1,j} = (j-1)q$, for $1 \leq j \leq m+1$

Theorem 1 is for computing the weights of minimum weight paths composed of only deletions or insertions.

<u>Theorem 2</u>: $D_{i,j} = \min (D_{i-1,j}+r, D_{i-1,j-1}+p, D_{i,j-1}+q)$
for $2 \leq i \leq n+1$, $2 \leq j \leq m+1$

The proofs of these theorems are trivial and can be found in Wagner's paper /66/ in different terminology. The above theorems provide a recursive structure of computation.

The uniform cost assignment uses the same weight for each edit operation regardless of the position of the edge in the path. In some applications of the WLD, the position of an edge in a path is not important, while in applying the WLD to grammatical inference the position of an edge in a path should be considered. The following example will show the difference between considering and not considering the position of an edge in a path.

<u>Example 3</u>: Let A : = (a+b), B : = (a), C : = ((a)). Based on the uniform cost assignment, we find that
$D(A,B) = 2r, D(C,B) = 2r$, and $D(A,C) = 2q+2r$.

When applied to Ex.3, the constructive procedure in Chap. 5 constructs grammars G_1, G_2, G_3 for strings A, B, C, respectively. Then, according to the dissimilarities of their languages, grammars are merged in a stepwise fashion. Since $D(A,B) = D(C,B) = 2r$ is the smallest dissimilarity, we either merge G_1 with G_2, or merge G_2 with G_3. If we merge G_2 with G_3, we may obtain a self-embedded production (Sec. 2.5.2.), but if we merge G_1 with G_2, we will not obtain a self-embedding production since A and B do not possess the substring property. To force the merger of G_2 with G_3 first, D(C,B) should be less than D(A,B). This can be accomplished if we consider the position of an edge in a path as a factor in assigning weight to the edge. If, in Ex. 3, we assume that any insertion or deletion made at an end of the string has less effect on syntactical structure than one made inside of the string, then D(C,B) can be made less than D(A,B).

In the rest of this section we will propose several modifications of the WLD. One of them adjusts the weight of the minimum weight paths found under the uniform weight assignment, and is discussed in Sec. 3.3.2. Other modifications adjust the edge weights in the digraph according to position and are presented in subsequent sections.

3.3.2. MODIFICATION I - PATH WEIGHT ADJUSTMENT

In this section, we propose a method that adjusts the weight of a minimum-weight path along with techniques for storing and retrieving paths. The idea is to subtract a certain amount of weight from the minimum weight measured under the uniform weight assignment. The weight of a minimal weight path is adjusted if the path does not contain non-zero weight change operations, and if insertion and deletion operations do not appear in the same path. The weights assigned to the insertions and deletions depend on position in the string. We assume that any edit operation occurring at the end of string has less effect than one occurring in the middle. The principle is to reduce the weight for those edit operations that occur before the first or after the last zero-weight change operation, as compared to those that occur at other positions.

Example 4: Let S = { ab, cb, acbc }, $p = 2$, $q = r = 1$. Let D, I, C, C' denote deletion, insertion, non-zero weight change and zero weight change operations, respectively. Then the dissimilarities and all minimal paths for each pair of strings are listed below.

a) D(ab,cb) = 2, $P_1 = s_{11}s_{21}s_{22}s_{33}$, $P_2 = s_{11}s_{12}s_{22}s_{33}$, $P_3 = s_{11}s_{22}s_{33}$ with corresponding operation sequences 0_1 = DIC', 0_2 = IDC', 0_3 = CC'. Since all minimal weight paths include both deletions and insertions, their weights are not candidates for adjustment.

b) D(ab,acbc) = 2, $P_1 = s_{11}s_{22}s_{23}s_{34}s_{35}$, with operation sequence 0_1 = C'IC'I. The weight of the right end insertion can be adjusted, and $D'(ab,acbc) = 2-e$, where $0<e<1$, is the unit of adjustment.

c) $D(cb,acbc) = 2$, $P_1 = s_{11}s_{12}s_{23}s_{34}s_{35}$, with operation sequence $0_1 = IC'C'I$. The weights of both insertion operations can be adjusted and $D'(cb,acbc) = 2-2e$. After adjusting the dissimilarities, we obtain

$D'(cb,acbc)<D'(cb,acbc)<D'(ab,acbc)<D(ab,cb)$.

We now formalize the procedure for ordering dissimilarities. Weights are assigned to the edges in a minimal weight path P consisting of all insertions or of all deletions, and zero-weights change operations only. The rules are as follows:

(1) For every insertion operation occurring before the first or after the last zero-weight change operation, assign weight q_1; for those occurring at other positions in P assign weight q_2, $0<q_1<q_2<q$.

(2) For every deletion operation occurring before the first or after the last zero-weight change operation assign weight r_1, and for those occurring at other positions in P assign weight r_2, $0<r_1<r_2<r$. The reason for reducing the weights for insertions and deletions occurring inside of a string is that we want to distinguish between mixed operations and pure operation sequences. A mixed operation sequence consists of more than one kind of operation besides the zero-weight change operation.

<u>Example 5</u>: Let A : = cb, B : = cab (Fig. 3). Then after adjusting the minimal path weight, $D'(A,B) = q_1 < q$.

A minimal weight path selected arbitrarily will not necessarily remain minimum after adjustment. The following example demonstrates the problem.

<u>Example 6</u>: The minimal weight paths from 'abb' to 'cab' are:

$P_1 = s_{11}s_{12}s_{23}s_{34}s_{44}$,

$P_2 = s_{11}s_{12}s_{23}s_{33}s_{44}$. Their corresponding operation sequences are 0_1 = IC'C'D, 0_2 = IC'DC'. After adjusting, the weight of P_1 will be q_1+r_1, which is less than q_1+r_2, the weight of P_2.

To assure that a minimal weight path will be the minimal weight path after adjusting, we have to check all minimal weight paths found in the digraph.

Let A and B be strings with length n and m, respectively. Under the uniform weight assignment, all minimal weight paths in the AB digraph can be stored in an (n+1) x (m+1) matrix. This matrix is equivalent to the matrix representing the AB digraph. Each entry in the matrix corresponds to a node in the AB digraph, and represents one or more of the three operations. Recall that $D_{i,j}$ is the minimal weight from s_{11} to $s_{i,j}$ in the AB digraph. The algorithm based on Theorems 1 and 2 applies no more than three operations to reach node $s_{i,j}$ from its nearest nodes. There can be several minimal weight paths from s_{11} to $s_{i,j}$. The last edit operation on any such path must be one of the three possibilities. The set of all possible last operations on minimal weight paths from s_{11} to $s_{i,j}$ will be stored in $L_{i,j}$ according to the following code:

code	1	2	3	4	5	6	7
operation	I	D	C	I&D	I&C	D&C	I&D&C

Example 7: Let A : = ab, B : = ac. Under the uniform cost assignment p = q+r, q = r, and the 3 x 3 matrix L is as follows:

$$L = \begin{bmatrix} 2 & 2 & 7 \\ 2 & 3 & 1 \\ 0 & 1 & 1 \end{bmatrix} = (L_{i,j})$$

The minimal weight paths can be found by tracing back indices in the array. Since L_{33} is 7, a minimal weight path exists having all three edit operations for reaching node s_{33}. Consider L_{32}, L_{22}, and L_{23}. Node s_{32} is reached by deletion, s_{22} is reached by change, and s_{23} is reached by insertion. Following the same rule, we continue tracing back to s_{11}. Thus we find the following minimal paths:

$P_1 = s_{11}s_{22}s_{32}s_{33}$, $P_2 = s_{11}s_{22}s_{23}s_{33}$, $P_3 = s_{11}s_{22}s_{33}$.

This procedure is different from the procedures proposed by Wagner and by Okuta et al. on two counts.

(1) The position of edges are considered in computing the weight of path.

(2) The minimal weight path is not found iteratively. All minimal weight paths are found under some cost assignment. Then all paths are adjusted according to the positions of edges in the path.

3.3.3. MODIFICATION II - NORMAL EDGE WEIGHTING

A second way of distinguishing between D(A,B) and D(C,B) in Ex. 3 is to vary the edge weights in the digraph according to the edge positions. Assign smaller weights for edges at the ends and larger weights for edges in the middle of a path. We propose a technique called normal edge weighting for this purpose. Normal edge weighting assigns weights according to the following rules. The idea is to follow a normal probability density, centered in the middle of the digraph.

Let $[N]$ denote the integer part of N.

Let $1 \leq i \leq n$, $1 \leq j \leq m$.

(1) Assign weight $(i+j-1)q$ to edges $(s_{i,j}, s_{i,j+1})$, and weight $(i+j-1)r$ to edges $(s_{i,j}, s_{i+1,j})$ for $(i+j) \leq [(m+n+3)/2]$.

(2) Assign weight $(m+n+2-i-j)q$ to edges $(s_{i,j}, s_{i,j+1})$ and $(m+n+2-i-j)r$ to edges $(s_{i,j}, s_{i+1,j})$ for $(i+j) > [(m+n+3)/2]$.

(3) Assign weight $(w_1 + w_2 + w_3 + w_4)/2$ to edges $(s_{i,j}, s_{i+1,j+1})$, where w_1, w_2, w_3, w_4 are weights for edges $(s_{i,j}, s_{i+1,j})$, $(s_{i,j}, S_{i,j+1})$, $(s_{i+1,j}, s_{i+1,j+1})$, and $(s_{i,j+1}, s_{i+1,j+1})$ respectively. If the ith character of

A is equal to the jth character of B then assign weight zero to edge $(s_{i,j}, s_{i+1,j+1})$.

The following example demonstrates normal edge weighting.

<u>Example 8</u>: Let A : = cb, B : = cab. Then the AB digraph and its normal edge weights are indicated in Fig. 4, and D(A,B) = 3q. All unnumbered edge weights equal 5(q+r)/2.

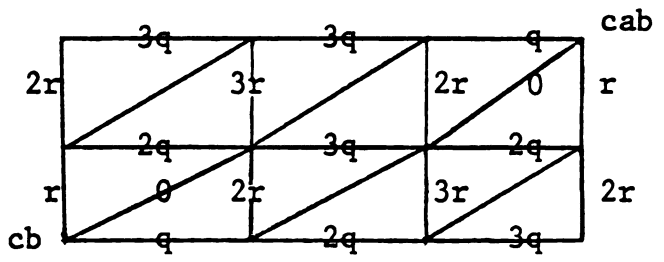


FIG.4. AB digraph under normal edge weighting

Let n, m be the lengths of strings A and B respectively, and q = r = 1. Then the normal edge weighting of the AB digraph has the following properties:

(1) The maximum weights for edges $(s_{i,j}, s_{i+1,j})$ and $(s_{i,j}, s_{i,j+1})$ are the same and equal $[(m+n+1)/2]$.

(2) The maximum weight of edge $(s_{i,j}, s_{i+1,j+1})$ is $(m+n)$.

(3) The maximum path weight is $(m+n+1)^2/4$, if m+n is odd; $(m+n)(m+n+2)/4$, if m+n is even.

Let $e = [(m+n+3)/2]$, and let $k = \min\{e,n\}$, and $k' = \min\{e,m\}$. Under the normal edge weighting, Theorems 1 and 2 become the following. The proofs of these theorems are analogous to the proofs of Theorems 1 and 2.

Theorem 3: $D_{i,1} = i(i-1)r/2$, for $1 \leq i \leq k$

$D_{i,1} = k(k-1)r/2 + (i-k)(2m+2n+3-i-k)r/2$, for $k < i \leq n+1$

$D_{1,j} = j(j-1)q/2$, for $1 \leq j \leq k'$

$D_{1,j} = k'(k'-1)q/2 + (j-k')\,(2m+2n+3-j-k')q/2$, for $k' < j \leq m+1$

Theorem 4: Let w_1, w_2, w_3 be weights of edges $(s_{i-1,j-1}, s_{i,j})$, $(s_{i-1,j}, s_{i,j})$, and $(s_{i,j-1}, s_{i,j})$, respectively. Then

$D_{i,j} = \text{Min}\,(D_{i-1,j-1} + w_1,\ D_{i-1,j} + w_2,\ D_{i,j-1} + w_3)$ for $2 \leq i \leq n+1$, $1 \leq j \leq m+1$.

The difference between normal edge weighting and WLD is that normal weighting assigns different weights to edges according to their position in the digraph, while the WLD assigns the same weight for all edges of the same type in the digraph.

3.3.4. MODIFICATION III - BINORMAL EDGE WEIGHTING

A third way of distinguishing between D(A,B) and D(C,B) in Ex.3 is to vary the edge weights in the digraph according to the row and column positions of the edge in the digraph and the type of operation. Assign smaller weights for edges at the ends of column (vertical direction) and row (horizontal direction) and larger weights for edges in the middle columns and rows. The idea is to distinguish between the same operation occurring in different places. We call this weighting binormal edge weighting since we assign normal weights to deletion operations and normal weights to insertion operations according to their positions.

Binormal edge weighting assigns weights to edges according to the following rules: Consider all i, j $1 \le i \le n$, $1 \le j \le m$,

(1) Assign weight (i)r to edge $(s_{i,j}, s_{i+1})$ for $1 \le i \le \lceil (n+1)/2 \rceil$; otherwise assign weight (n+1-i)r to edge $(s_{i,j}, s_{i-1,j})$.

(2) Assign weight (j)q to edge $(s_{i,j}, s_{i,j+1})$ for $1 \le j \le \lceil (m+1)/2 \rceil$; otherwise assign weight (m+1-j)q to edge $(s_{i,j}, s_{i,j+1})$.

(3) Let w_i, w_j be the weight of edges $(s_{i,j}, s_{i+1,j})$ and $(s_{i+1,j}, s_{i+1,j+1})$ respectively. If the ith character of A is equal to the jth character of B then assign weight w_i+w_j to the edge $(s_{i,j}, s_{i+1,j+1})$. The following example demonstrates binormal edge weighting.

<u>Example 9</u>: Let A : = cb, B : = cab. Then the AB digraph and its binormal edge weights are indicated in Fig. 5, and D(A,B) = 2q.

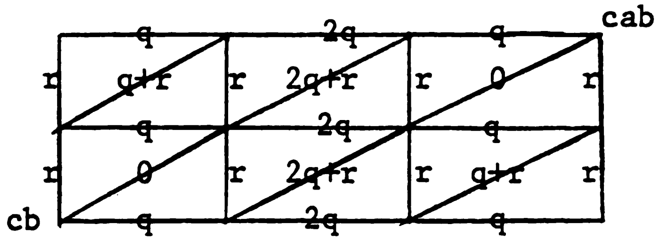


FIG.5. AB digraph under binormal edge weighting

Let n, m be the lengths of strings A and B respectively, and q = r = 1. Then the binormal edge weighting has the following properties:

(1) The maximum weight of edge $(s_{i,j}, s_{i+1,j})$, $1 \leq i \leq n$, $1 \leq j \leq m+1$ is $\lceil (n+1)/2 \rceil$.

(2) The maximum weight of edge $(s_{i,j}, s_{i,j+1})$, $1 \leq i \leq n+1$, $1 \leq j \leq m$ is $\lceil (m+1)/2 \rceil$.

(3) The maximum weight of edge $(s_{i,j}, s_{i+1,j+1})$ is $(m+n)/2$, if n, m are both even; otherwise $\lceil (m+n+2)/2 \rceil$.

(4) The maximum path weight is $((m+1)^2 + (n+1)^2 - 2)/4$ if n, m are both even; $((n+2)^2 + (m+1)^2)/4$ if n, m are both odd; $((m+1)^2 + (n+2)^2 - 1)/4$ if one is even and one is odd.

Let $k = \lceil (n+3)/2 \rceil$, and $k' = \lceil (m+3)/2 \rceil$. Under the binormal edge weighting, Theorems 1 and 2 become the following. The proofs of these theorems are analogous to the proofs of Theorems 1 and 2.

Theorem 5: $D_{i,1} = i(i-1)r/2$, for $1 \leq i \leq k$

$D_{i,1} = k(k-1)r/2 + (i-k)(2n+3-i-k)r/2$, for $k < i \leq (n+1)$

$D_{i,j} = j(j-1)q/2$, for $1 \leq j \leq k'$

$D_{1,j} = k'(k'-1)q/2 + (j-k')(2m+3-j-k')q/2$, for $k' < j \leq (m+1)$

Theorem 6: Let w_1, w_2, w_3 be the weights of edges $(s_{i-1,j-1}, s_{i,j})$, $(s_{i-1,j}, s_{i,j})$, and $(s_{i,j-1}, s_{i,j})$, respectively. Then

$D_{i,j} = \text{Min}\,(D_{i-1,j-1}+w_1, D_{i-1,j}+w_2, D_{i,j-1}+w_3)$

for $2 \leq i \leq n+1$, $2 \leq j \leq m+1$.

3.3.5. MODIFICATION IV - DEPTH EDGE WEIGHTING

A fourth way of distinguishing between D(A,B) and D(C,B) in Ex.3 is to vary the edge weights in the digraph according to the number of steps from the source node. Assign smaller weights to low frequency edges and larger weights to high frequency edges, where frequency is defined below. The idea is to distinguish operations by their positions in the digraph with a depth measure so this weighting is called depth edge weighting.

Excluding the diagonal edges, the AB digraph has $T = (2mn+m+n)$ edges. Let h' be an integer function defined as $h'(i,j) = (i-1) + (j-1)$ for all i, j, $1 \le i \le n+1$, $1 \le j \le m+1$. $h'(i,j)$ is the number of steps needed to reach $s_{i,j}$ from s_{11}, not allowing diagonal steps. Furthermore, let f'_k denote the number of non-diagonal edges which terminate on nodes $s_{i,j}$ for which $h'(i,j) = k$. Then, depth edge weighting assigns weights to edges according to the following rules:

(1) Assign weights f'_k/T to edges $(s_{i,j}, s_{i+1,j})$, for $1 \le i \le n$, $1 \le j \le m+1$, and to edges $(s_{i,j}, s_{i,j+1})$, for $1 \le i \le n+1$, $1 \le j \le m$.

(2) Let w_1, w_2 be the weights of edges $(s_{i,j}, s_{i+1,j})$ and $(s_{i+1,j}, s_{i+1,j+1})$ for all i, j $1 \le i \le n$, $1 \le j \le m$. If the ith character of A is not equal to the jth character of B, then assign weight $w_1 + w_2$ to the edge $(s_{i,j}, s_{i+1,j+1})$. Otherwise assign weight zero to the edge.

Example 10: Let A : = cb, B : = cab be strings, n = 2, m = 3.

The number of non-diagonal edges in the AB digraph is T = 17. The function value for h' and f'_k are:

$$h'(i,j) = \begin{array}{c|cccc} i \backslash j & 1 & 2 & 3 & 4 \\ \hline 3 & 2 & 3 & 4 & 5 \\ 2 & 1 & 2 & 3 & 4 \\ 1 & 0 & 1 & 2 & 3 \end{array}$$

k	f'_k
1	2
2	4
3	5
4	4
5	2

The edges terminating at nodes two steps from A_{11} are:

(s_{21},s_{31}), (s_{21},s_{22}), (s_{12},s_{22}), (s_{12},s_{13})

Thus $f'_2 = 4$ and each edge weight is $f'_2/T = 4/17$.

The depth edge weights in Fig. 6 are the number shown divided by 17, and D(A,B) = 5/17.

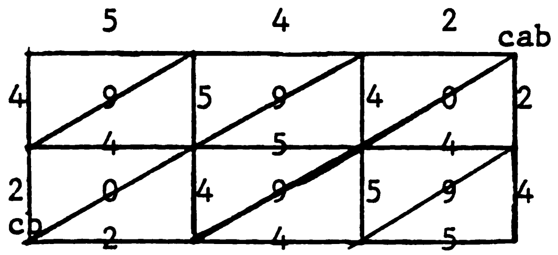


FIG.6. AB digraph under depth edge weighting.

Let k = min {n,m}. The depth edge weights of the AB digraph have the following properties:

(1) The maximum weights for edges $(s_{i,j},s_{i+1,j})$ and $(s_{i,j},s_{i,j+1})$ are the same and equal (2k+1)/(2mn+m+n).

(2) The maximum weight of a diagonal edge is (4k + 2)/(2mn+m+n), n+m is even; otherwise, (4k+1)/(2mn+m+n).

(3) The maximum weight of a path is 1.

Depth edge weighting depends heavily on the lengths of strings. Actually, depth edge weighting classifies edges into three categories. The weights of edges on both ends of a path are varied gradually, while edges in the middle of a path have the same weight.

Under depth edge weighting, Theorems 1 and 2 become the following. The proofs of these theorems are analogous to the proofs of Theorems 1 and 2.

Theorem 7: $D_{i,1} = i(i-1)/T$, for $1 \le i \le (k+1)$

$D_{i,1} = (k(k+1) + (i-k-1)(2k+1)/T$, for $(k+1) < i \le n+1$

$D_{i,j} = j\ (j-1)/T$, for $1 \le j \le (k+1)$

$D_{1,j} = (k(k+1) + (j-k-1)(2k+1))/T$, for $(k+1) < j \le m+1$

Theorem 8: Let w_1, w_2, w_3 be weights of edges $(s_{i-1,j-1}, s_{i,j})$, $(s_{i,j-1}, s_{i,j})$, and $(s_{i-1,j}, s_{i,j})$, respectively. Then

$D_{i,j} = \text{Min}\ (D_{i-1,j-1}+w_1, D_{i,j-1}+w_2, D_{i-1,j}+w_3)$ for $2 \le i \le n+1$, $2 \le j \le m+1$.

The dissimilarity measures presented in this section will be used in the cluster analysis of the syntactical structure of sample strings, and will be evaluated by the complexity of the resultant grammar (see Chap.IV). Example 11 demonstrates the differences among the dissimilarity measures proposed in this section.

Example 11: Let A : = (a+b), B : = (a), C : = ((a)). Then

(1) Under uniform edge weight, Ex. 3, (U_w)

$D(A,B) = D(C,B) = 2r$, $D(A,B)/D(C,B) = 1$

(2) Under adjusted path weight, (A_w)

$D(A,B)/D(C,B) = r_2/r_1$

(3) Under normal edge weight, (N_w)

$D(A,B) = 7r$, $D(C,B) = 2r$,

$D(A,B)/D(C,B) = 7/2$

(4) Under binormal edge weight, (B_w)

$D(A,B) = 5r$, $D(C,B) = 2r$,

$D(A,B)/D(C,B) = 5/2$

(5) Under depth edge weight, (D_w)

$D(A,B) = 13/38$, $D(C,B) = 4/38$,

$D(A,B)/D(C,B) = 13/4$

Assume $1<r_2/r_1<\frac{3}{2}$. Then the ratios are ordered as:

$U_w<A_w<B_w<D_w<N_w$

Example 11 demonstrates that normal edge weighting best differentiates self-embedding from inner substring relations. Example 12 indicates that uniform edge weighting and adjusted path weighting do not differentiate self-embedding from left or right substring relations, while normal, binormal and depth edge weightings are equivalent in this task.

Example 12: Let $A :=$ abab, $B :=$ ab, $C := a^2b^2$

(1) Under U_w, $D(A,B) = D(C,B) = 2r$,

$D(A,B)/D(C,B) = 1$

(2) Under A_w, $D(A,B) = D(C,B) = 2r_1$,

$D(A,B)/D(C,B) = 1$

(3) Under N_w, $D(A,B) = 3r$, $D(C,B) = 2r$

$D(A,B)/D(C,B) = 3/2$

(4) Under B_w, $D(A,B) = 3r$, $D(C,B) = 2r$

$D(A,B)/D(C,B) = 3/2$

(5) Under D_w, $D(A,B) = 6/22$, $D(C,B) = 4/22$

$D(A,B)/D(C,B) = 3/2$

The ratios are ordered as:

$U_w = A_w < B_w = D_w = N_w$

3.4. CLUSTER ANALYSIS

The syntactic relations among strings of a given sample are analyzed to provide information for constructing an efficient grammar for that sample. In this section, clustering methods are used to impose an hierarchical structure on the sample.

The problem of sorting similar things into categories is known as the category sorting problem. Clustering algorithms and their use in pattern recognition have been surveyed /39/, and discussions of several techniques and listings of computer programs for implementing them have been provided /1/,/17/.

Cluster analysis is the key step in the category sorting problem. In cluster analysis, little or nothing is known about the data structure. The essence of cluster analysis is to assign appropriate meaning to the terms "natural groups" and "natural association". The results of a cluster analysis can contribute directly to development of classification schemes, and can be used to develop inductive generalizations. A general discussion of cluster analysis can be found in Anderberg /1/.

Johnson /38/ argues that good clustering algorithms should satisfy the following three properties:

(1) Input data should consist solely of a point set or a matrix of similarities.

(2) The method should be such that a clear, explicit and intuitive description of what the clustering accomplishes is possible.

(3) The method should be invariant under monotone transformations on the similarity measure.

Clustering methods based on the minimal spanning tree (MST) /69/ satisfy most of these principles. A set of sample strings can be treated as a set of points in an abstract space and dissimilarities can be measured adequately as discussed in Sec. 3.3. For these reasons, the MST will be the first tool for analyzing a set of sample strings. In this section, we will review a clustering method based on the MST and discuss the problem of forming clusters from the labelled MST.

3.4.1. LABELLED MINIMAL SPANNING TREE

In this section, we will define the labelled MST for a complete graph whose edge weights are given in a dissimilarity matrix. Each node in the graph represents a sample string.

A lexicographic order of strings is defined as follows:

Definition: Let S be a set of sample strings with V_t the terminal alphabet. If V_t is given an arbitrary fixed order, then a unique lexicographic order for S can be defined by the following rules:

(1) For x, $y \in S$, if $\ell(x) < \ell(y)$, then x precedes y in S, where $\ell(x)$ is the length of x.

(2) For any strings $x = a_1 \ldots a_n$ and $y = b_1 \ldots b_n$ for which $\ell(x) = \ell(y) = n$ and $a_i = b_i$ for $1 \leq i \leq k$ and $a_{k+1} \neq b_{k+1}$, $0 \leq k \leq n-1$, then x precedes y in S if a_{k+1} precedes b_{k+1} in V_t.

Example 12: Let V_t = (a,b,c), be the set of ordered terminals. Then set S is in lexicographic order;

L_s = (cb, bbb, cab, baab, bbab, bbaab, caaab).

An undirected graph is complete if there is an edge connecting each pair of nodes directly. An n by n dissimilarity matrix defines the edge weights for a complete graph with n nodes and n(n-1)/2 edges. The (i,j) entry in the matrix is the weight of the edge connecting the nodes corresponding to strings i and j. A spanning tree is any set of n-1 edges which provides one and only one path between any pair of nodes. The weight of a spanning tree is the sum of the weights for edges in the tree. A minimal spanning tree is a spanning tree having minimum weight among all spanning trees.

A <u>labelled MST</u> for a set of sample strings is a MST on which a Gorn tree structure /41/ defined below has been imposed. The root corresponds to the first string in the lexicographic order. The immediate successors to the root are those nodes in the MST adjacent to the root. These nodes are at level 1 and are arranged in the lexicographic order of the corresponding strings. Similarly, let N be a node at level i. All nodes in the MST adjacent to N not already in the tree structure are placed at level i+1 and are written in the lexicographic order of the corresponding strings.

<u>Example 14:</u> Let S = {a,ab,abc,(a),(b) }. Under uniform edge weighting the dissimilarity matrix is shown below and a labelled MST is given in Fig. 7.

	ab	abc	(a)	(b)
a	1	2	2	4
ab		1	3	3
abc			4	4
(a)				2

dissimilarity matrix

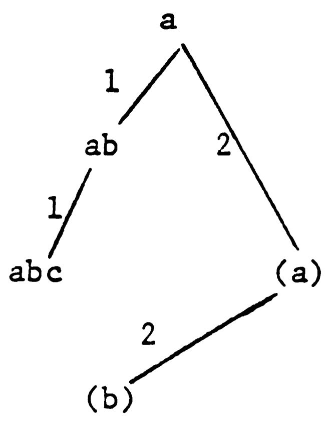


FIG. 7. Labelled MST

There are many powerful algorithms for finding a MST /42/,/45/,/53/. One of the most popular algorithms was given by Prim /53/. Clustering algorithms based on the MST have been developed /18/,/31/,/55/,/60/.

3.4.2. CLUSTERING METHOD

In this section we will review an hierarchical clustering method suggested by Zahn /69/. The clustering at level d_i can be obtained from the MST by deleting all edges of length greater than d_i. Each connected subgraph represents a cluster. The single link dendrogram can also be derived from the MST /37/,/38/. Clustering methods based on the MST delete edges, called inconsistent edges, from the MST so that the resulting connected subtrees correspond to meaningful clusters. This problem is similar to cutting a dendrogram to form clusters.

An edge XY in a MST whose weight W(XY) is significantly larger than the average of nearby edge weights on both sides of edge XY can be called an inconsistent edge /69/. There are two natural ways to measure inconsistency. The first type of inconsistency uses edge weight standard deviations. An edge XY is inconsistent if W(XY) differs by a few standard deviations from the average edge weight. Inconsistency of this kind is dependent on the following factors:

(1) The size of the neighborhood explored for each end of the edge XY;

(2) The number of standard deviations and the factor considered as significant;

(3) Whether or not inconsistency is required at both ends of the edge XY.

If the distribution of edge weights in the MST is normal, then an edge weight exceeding the mean by three or four standard deviations would occur less than one percent of time and hence may be regarded as significant.

A second type of inconsistency uses the factor or ratio between W(XY) and the average of neighboring edge weights. A factor of 2 or more, usually means the separation is quite apparent. The edge inconsistency defined by factor also depends on (1) and (3) above.

The type of edge inconsistency based on standard deviation has statistical distributional significance while inconsistency based on factor or ratio has intuitive significance. These are only two possibilities. From a practical point of view, edge inconsistency should depend on the source of the data, the dissimilarity measurement and the specific purpose of the cluster analysis. In grammatical inference, neither type of inconsistency described previously may be able to provide a complete satisfactory result for two reasons.

(1) The dissimilarity measure on strings does not take all structural relations between strings into account.

(2) The assumptions made on the given set of sample strings are not sufficient to decide which edge inconsistency definition is appropriate.

For these reasons, structural relations other than those reflected in the dissimilarity matrix may be needed for inferring a grammar. The following example will show one such structural relation.

Example 15: Let S = {abab, aba, ab, a, ac, acc, accc }. Let p = 2, q = r = 1. The dissimilarity matrix under uniform edge weighting is given below. A MST links the strings with edges of length one in the order: abab, aba, ab, a, ac, acc, accc.

	ab	ac	aba	acc	abab	accc
a	1	1	2	2	3	3
ab		2	1	3	2	4
ac			3	1	4	2
aba				4	1	5
acc					5	1
abab						6

Only one cluster is formed from S in Ex.15 under single-link clustering. Intuitively, there are two different sequences of strings in S. Adapting different dissimilarity measures will not change the number of clusters in Ex.15. If S can be split into two groups {abab, aba, ab} and {a, ac, acc, accc}, then two small grammars can be constructed. This procedure would be simpler than constructing one grammar for S, since S does not show a clear picture of sample structure. How can S be split into two groups based on the MST? The problem is to define an inconsistent edge in the MST. We need to know what kind of clusters we are expecting, and the characteristic of the inconsistent edge. Intuitively, the length of a string can be considered as a factor. When the edges in a MST have uniform length, the edges adjacent to the node or nodes representing the shortest string or strings will be considered as inconsistent edges. In Ex.15 the edges (ab,a) and (a,ac) are inconsistent edges. At the first step of cluster analysis, we can delete either of them. If we delete the edge (ab,a) then we obtain two clusters {abab, aba, ab}, and {a, ac, acc, accc}. In the next section, we will discuss the problem more carefully. Other definitions of edge inconsistency will also be given.

3.4.3. CLUSTERING METHOD FOR GRAMMATICAL INFERENCE

One major objective of this thesis is to find recursive rules for a language based on a finite set of sample strings. Since a finite set of sample strings contains only limited information, recursive structures may not be inherent in the sample strings. One tactic is to search for all potential recursive structures in the MST. A potential recursive structure can be found from a sequence of strings whose lengths gradually increase. Actually, two factors should be considered: "length" and "common symbols". If a sequence of strings has all the following properties, then there is a potential recursive structure.

Let $\{x_i\}$ be a sequence of strings. Then

(1) for all i, $\ell(x_i) + k = \ell(x_{i+1})$, where k is a positive integer;

(2) for all i, x_i is a substring of x_{i+1};

(3) the character concatenation difference between x_i and x_{i+1} is constant for all i.

These properties are compatible with all dissimilarity measures defined in Chap. III. When a sequence of strings satisfies these properties, the dissimilarity between successive pair of strings is constant.

In grammatical inference by constructive methods, we intend to infer a grammar with a small number of productions and an adequate grammatical framework at the beginning of construction. Also, we want to discover all proper recursive structures for the sample. Thus, the clustering method for grammatical inference will be different from conventional clustering methods. These goals can be achieved by generating an inference tree defined below.

An inference tree is a labelled tree defined on a labelled MST by a partition function. The inference tree is most significant for the constructive methods for generating grammars given in Chap. V. Let $T = \{T_1, T_2, \ldots, T_n\}$ be a partition of a labelled MST. Each $T_i \in T$ represents a labelled subtree. The index i of each subtree $T_i \in T$ is assigned by an index function associated with the partition function. Then the inference tree over T is defined as:

(1) The root corresponds to the subtree T_1 of index 1.

(2) The immediate successors to the root correspond to those subtrees in T whose roots are adjacent to T_1 in the labelled MST. These nodes are at level 1 and are arranged in the descending order of the corresponding subtree indexes.

(3) Similarly, let N be a node at level i. The immediate successors to the node N correspond to those subtrees in T whose roots are adjacent to the subtree associated with node N. These nodes are placed at level i+1 and written in descending order of the corresponding subtree indexes.

The clustering algorithm proposed, based on a labelled MST for obtaining an inference tree is described below. The actual procedures are given in the Appendix. The clustering algorithm starts at the root of the labelled MST, follows breadth first search to examine the relationship between the string corresponding to the root and the strings corresponding to its immediate successors. Two kinds of relationships, one based on "length" and the other based on "common substring", are expressed as seven conditions. These conditions are listed below and applied in sequence to delete edges connecting the root, or the node in question and its immediate successors. After a complete application of the conditions to all edges between the node and its immediate successors, at most one edge will remain. Also, whenever an edge between a node N and an immediate successor M of N is disconnected, M becomes a new root. Let all new roots be numbered in the order of becoming new roots.

The search starts at the initial root of the labelled MST. The algorithm repeats the same process to classify subtrees, according to the descending order of the number associated with the root of subtrees. The algorithm ends when no more subtrees can be formed. See Appendix.

Let k be the maximal length of common strings, u be the minimal number of distinct symbols, and d be the minimal dissimilarity between the string corresponding to a node N and all strings corresponding to its undisconnected immediate successors. The seven conditions for inconsistency edges, in the order of application, are listed below. Let edge (N,M) be the edge under investigated, and let N' be the immediate predecessor of N.

(1) N and M have the same length;

(2) The lengths of N' and M are both greater than or less than the length of N;

(3) The length of maximal common substring to N and M is zero;

(4) The length of maximal common substring to N and M is less than k;

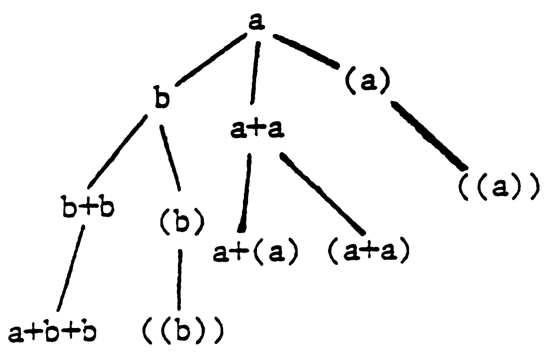
(5) The number of distinct symbols not being used in both N and M is greater than u;

(6) The edge weight of the edge (N,M) is greater than d;

(7) There exists an undisconnected immediate successor of N preceding M in lexicographic order.

<u>Example 16:</u> Let $V_t = \{a,b,+,(,)\}$ be an ordered terminal set, and let S = {a, b, a+a, b+b, (a), (b), a+b+b, a+(a), (a+a), ((a)), ((b))} be a set of sample strings. Under a uniform edge weighting, a labelled MST for S and the labelled tree resulting from the clustering algorithm in the Appendix are shown in Fig.8. All edges in the labelled MST have length 2.

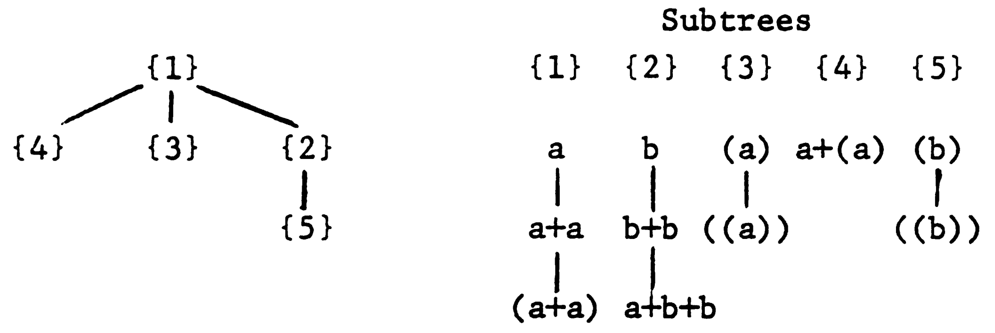
	b	a+a	b+b	(a)	(b)	a+b+b	a+(a)	(a+a)	((a))	((b))
a	2	2	4	2	4	4	4	4	4	6
b		4	2	4	2	4	6	6	6	4
a+a			4	4	6	4	2	2	6	8
b+b				6	4	2	6	6	8	6
(a)					2	6	6	2	2	4
(b)						6	4	4	4	2
a+b+b							6	6	8	8
a+(a)								2	4	6
(a+a)									2	6
((a))										2



(a) A labelled MST for S

FIG. 8. Labelled MST and inference tree for Ex.16.

FIG. 8. (Continued . . .)



(b) An inference tree for Ex.16

3.5. SUMMARY

This chapter concentrates on the analysis of the syntactical structure of a set of sample strings. Methods for analyzing sample structure are reviewed, and ways of using cluster analysis to suggest sample structure are proposed.

Dissimilarity measures between strings based on the sequence of edit operations are presented, and the problem of assigning weights to edges of the AB digraph are discussed. Four alternative assignments of edge weights are proposed and their properties are studied. Algorithms for computing dissimilarity matrix and procedures for adjusting edge weights are proposed.

The MST concept and clustering methods based on the MST are reviewed. The disadvantages of using conventional clustering methods to classify a set of sample strings for the purpose of grammatical inference are discussed. A labelled MST for a set of sample strings is defined, and an hierarchical clustering algorithm based on the labelled MST is proposed. The resulting clusters related by an inference tree should provide insight, perhaps sufficient information for constructing a grammar.

The clustering method is different from the conventional clustering method in three aspects.

(1) The clustering method uses information which is not given by the dissimilarity matrix, such as "length" and "common symbols".

(2) The clustering method is based on a labelled MST and uses breadth first search to delete inconsistent edges.

(3) The clusters obtained by this method are labelled subtrees, which are related by an inference tree.

CHAPTER IV

GRAMMATICAL COMPLEXITY AND ACCEPTANCE CRITERIA

4.1. INTRODUCTION

If a problem is to have meaning, it is necessary to define what is meant by a "solution to the problem". For instance, the solution to an algebraic problem is usually definite and unique, while the solution of a grammatical inference problem is not. In fact optimality cannot be practically determined.

In inferring a grammar for a set of sample strings, it is very important to specify the type of solution grammar desired. Without specifying the characteristics of the solution, the inference procedure will never know when to stop. The type of solution grammar chosen indirectly defines the inference procedure and implies a stopping rule.

What are the necessary characteristics of the solution grammar? This question can be answered in a number of ways. The solution grammar can be selected with the particular inference procedure adopted in mind. Several disparities exist in the solution grammars that can be obtained with enumerative and constructive methods. Even among enumerative methods, the assumption concerning information presentation and the assumption about the teacher in the learning process will generate different solution grammars. In inferring grammars by constructive methods, the solution grammar is often defined as the best grammar among a class of candidate grammars. This is the approach taken in this thesis. The "best" grammar is that which satisfies certain conditions. Frequently, the best grammar is defined in terms of grammatical complexity and language discrepancy /11/,/46/. The former refers to the complexity of the inferred grammar, the later refers to the discrepancy between the language generated by the inferred grammar and the given set of sample strings. The grammar which has the least complexity and/or smallest discrepancy among all candidate grammars is called the best grammar.

In Sec. 4.2., we will define solution grammars for the grammatical inference problem. A measure of grammatical complexity will be defined in Sec. 4.3. A difference measure between languages will be given in Sec. 4.4. A statistical method and an acceptance criterion for testing an inferred grammar will be proposed in Sec. 4.5.

4.2. SOLUTION GRAMMAR

In inferring grammars by constructive methods from a set of sample strings, the class of solution grammars is specified by grammatical properties, such as the type, the language generating capacity, and the complexity. Grammatical types are the well-known finite-state, context-free, context-sensitive, and universal types. Only finite-state and subclasses of context-free grammars have been studied in the field of grammatical inference. Language generating capacity refers to the quality of languages generated by the grammar. The difference between the language generated and the given set of sample strings may be measured and tested statistically as explained in Sec. 4.5. The results obtained from statistical tests will reflect the quality of the language. The complexity of a grammar involves both intrinsic complexity and derivational complexity (Cf. Sec. 4.3) /21/. In some research, only one of them is considered /59/. The complexity of a grammar can also be employed as a factor in evaluating the quality of an inferred grammar.

In this section, we will discuss the definition of solution grammar, and establish a new type of solution grammar.

4.2.1. HALTING PROBLEM

The halting problem arises when using enumerative methods in grammatical inference. Gold /11/ studied the relationship between the enumerative method and the type of information representation, and found that the time needed for an enumerative procedure to find the correct grammar depends on the manner of presenting information. Feldman /21/ and Wharton /59/ also studied the enumerative method based on grammatical complexity, and found that there exists an effective procedure to enumerate a class of grammars according to the descending order of their complexities, as long as there is a finite number of grammars for any given complexity.

The halting problem also arises in using constructive method to infer grammars. Here, the halting time depends on the constructive procedure and the solution grammar. With a constructive method, the solution grammar is often defined as the best grammar that can be obtained by the procedure. In other cases, the solution grammar is chosen to meet certain a-priori qualifications. A constructive method will stop when it obtains the solution grammar, or when it finds that obtaining the solution grammar is impossible.

How is the effectiveness of a constructive procedure measured? In order to answer this question, we have to examine the properties of the solution grammar which indirectly affect the construction process. Since the type of grammar to be inferred is determined by the source of sample strings, and the language generating capacity is used to test whether the inferred grammar is acceptable or not, the halting time of a constructive procedure will depend on grammatical complexity and will halt in a finite time if only a finite number of grammars can be constructed for any complexity. Therefore, the key problem is to define a measure of grammatical complexity that assigns the same complexity to only a finite number of grammars. We will discuss this problem later in this chapter.

4.2.2. RELATIONSHIP BETWEEN SAMPLE STRINGS AND GRAMMARS

There is no one-to-one relationship between a language and a grammar. Many grammars can generate a given language. Although a language may be identified with a grammar characterized by type and complexity, there are still many candidate grammars for a language. Identifying a grammar for a set of sample strings with a grammar is even more difficult, since there is an enormous number of finite and infinite languages containing that sample.

In inferring probabilistic grammars for a probablistic sample, the chi-square goodness of fit test has been employed to test the difference between the frequency distributions of the sample and the language generated by an inferred grammar. The Bayesian decision rule has also been used to infer a probabilistic grammar. A number of measurements based on string probability (Sec.2.3.) have been proposed which measure the probabilistic difference between a set of sample strings and a language.

No statistical techniques have been proposed for inferring non-probabilistic grammars, since no appropriate characteristics of non-probabilistic languages can be quantized. However, the possibility of using statistical techniques for decision making exists. Measurements other than string probability might also be developed.

4.2.3. THE DEFINITION OF SOLUTION GRAMMAR

In this thesis, the solution grammar will either be a finite-state grammar or a context-free grammar in Chomsky normal form which satisfies the following conditions:

(1) The grammar has the capacity to generate the set of sample strings;

(2) The language generated by the grammar is statistically close to the set of sample strings;

(3) The grammar is the simplest and least complex among candidate grammars.

The solution grammar for a set of sample strings should be the "best" grammar which the inference procedure can produce. Statistical tests and complexity measures characterize the solution grammar.

4.3. COMPLEXITY OF GRAMMARS

Grammatical complexity has been the basis of grammatical inference /11/,/24/,/36/,/67/ and may also be employed to define an inference procedure /24/,/67/. The function of grammatical complexity has been presented in Sec. 4.2. The general concept of grammatical complexity measures has been presented in Sec. 2.4. In this section, grammatical complexity measures and their applications to grammatical inference will be discussed. Size measure of grammar will be discussed in Sec. 4.3.1.-4.3.3. and used to evaluate the performance of different dissimilarity measures presented in Chapter III.

A complexity measure based on information theory will be defined in Sec. 4.3.4. for selecting the best grammar for the sample strings.

4.3.1. SIZE COMPLEXITY MEASURE

There are at least four different approaches to the notion of size measure for grammars (Cf. Sec. 2.4.1.). We will describe them briefly in this section.

Gruska /32/ classifies grammars in a class C by mappings from C into nonnegative integers. The classifications of languages correspond to those grammars. The intrinsic structure of a grammar G is characterized by the number and the depth of the grammatical levels. The grammatical level G_o of G is a maximal set of productions of G, the left-side symbols of which are mutually dependent in that the productions are chained together. This complexity measure is too broad since it does not discriminate among all intuitively desirable measures of complexity.

Blum /5/ defines the size measure of grammars in terms of the "bigness" of a grammar, referring to the number of nonterminals, terminals and productions. Preliminary notations are needed for understanding his measure.

Two grammars G_1 and G_2 are <u>completely equivalent</u> if one may be transformed into the other by one-to-one, onto mapping of their nonterminal vocabularies. For any grammar $G = (V_n, V_t, R, A)$ in a class of grammar C, the <u>complete equivalence class</u> of G is the set of grammars in C completely equivalent to G.

A complexity measure on a class C of grammars is a mapping from C into the nonnegative integers for which:

(1) There exists at most a finite number of complete equivalence classes of grammars of any complexity;

(2) There exists an effective procedure for determining which grammars are of complexity c for any c.

Under Blum's definition, a complexity measure which is valid for one class of grammars may not be valid for a larger class of grammars.

Wharton /32/ modifies the size measure by considering the maximum length of the right-hand side of a production. Let m(G) be the maximum length of the strings on the right-hand side of the productions of a grammar G, and n be the counting function. He adds the following two axioms to the above complexity measure.

(3) All grammars in C with the same value of m(G) and n(R) have the same complexity.

(4) Increasing either m(G) or n(R) without changing the other increases complexity.

4.3.2. GENERAL COMPLEXITY MEASURE BASED ON SIZE MEASURE

Feldman /21/ defined a general grammatical complexity measure. A sequence $<y_1,y_2,...>$ is said to be approximately ordered by a function f(y) if and only if there is a function $h_f(i)$ such that for each $i>1, t>h_f(i)$ implies $f(y_t) > f(y_i)$. If $h_f(i)$ is effectively computable, then $<y_1,y_2,...>$ is said to be effectively approximately ordered (EAO) by f, and f is said to be EAO by $< y_1, y_2, \ldots >$.

Let $\mathcal{Z}$ be the set of all finite sets S contained in V_t^*, and C be the class of grammars that generate $\mathcal{Z}$. Then a general complexity measure is a mapping f from $\mathcal{Z} \times C$ into the set of nonnegative rational members, satisfying the following conditions.

(C1) The function f is expressible in terms of the intrinsic complexity c(G,C) and the derivational complexity d(S,G) and is a computable unbounded increasing function of its two arguments.

(C2) The intrinsic complexity c(G,C) is a positive computable unbounded function EAO by the length of grammar.

(C3) The derivational complexity d(S,G) is a positive function and defined if and only if $S \in L(G)$.

(C4) There exists a computable function D(S,G,m) which is equal to zero if and only if $d(S,G) \leq m$, and 1 otherwise.

Feldman's intrinsic grammatical complexity is similar to Blum's size measure /5/,/6/. Instead of C2, Blum's complexity measure requires that an algorithm exists for computing the finite number of grammars with any fixed complexity. Wharton /67/ defines a complexity measure based on Blum's measure and requires a condition similar to C2.

Feldman's complexity measure is more restrictive than Blum's and Wharton's complexity measures, but it is still a general complexity measure. Many intuitive complexity measures will meet these requirements.

Solomma /56/ defined an index of a context-free grammar which is analogous to the derivational complexity. Since the relationship between the number of productions and the index of grammar is undecidable, the index of a context-free grammar will not meet the requirements for evaluating grammars.

4.3.3. COMPLEXITY MEASURE FOR EVALUATION

A simple complexity measure of grammars is needed to compare the sizes of grammars constructed from a cluster analysis on a set of sample strings. The general complexity measure given by Blum and modified by Wharton, meets most requirements. The axioms given by Blum for his complexity measure effectively enumerate a class of grammars, which is not necessary for evaluating a set of grammars. We now modify and restate the size measure of grammars as follows:

Let C be a set of grammars $G = (V_n, V_t, R, A)$. Then a general complexity measure on C is a mapping f from C into nonnegative real numbers, which satisfies the following axioms, where $n(\cdot)$, $m(\cdot)$ are defined as before.

(1) f is a positive unbounded function of $n(V_n)$, $n(V_t)$, $n(R)$ and $m(G)$.

(2) the effect of $m(G)$ on the function value is expontential.

(3) increasing any arguments increases complexity.

Example 1: Let C be a set of context-free grammars. Then $f(G) = n(V_n) + n(V_t) + n(R) + e^{m(G)}$ is a complexity measure on C.

Example 2: Let C be a set of Chomsky normal form grammars. Then $f(G) = n(V_n) + n(V_t) + n(R)$ is a complexity measure on C.

The number of productions which can be constructed with the right hand side of each production having length less than or equal to m(G) increases exponentially with the number of nonterminals. Therefore it is reasonable to assume that the contribution of m(G) is expontential.

4.3.4. COMPLEXITY MEASURE FOR SELECTING THE BEST GRAMMAR

We will adopt complexity measures based on information theory to select the best grammar for a set of sample strings. There are at least three different approaches to such complexity measures (Cf. Sec. 2.4.2.). Since the inference procedure proposed for this research is similar to Cook's hill-climbing method, we will adopt the complexity measure given by Cook in the sequel /11/.

The complexity of a grammar can be measured by the information required to specify that grammar. The information required to specify a grammar can be determined in three different situations. If a probability distribution over a class of grammars C is given, then the information conveyed by the selection of any particular grammar G is $-\log \Pr(G|C)$. In the second situation, we need a preliminary definition of a grammar-grammar, which is a grammar for generating grammars.

Definition: A grammar-grammar $\overline{G} = (\overline{V}_n, \overline{V}_t, \overline{R}, \overline{A})$ on the terminal alphabet V_t is defined to be a context-free grammar such that

(1) $V_n \cap W = \emptyset$

(2) $\overline{V}_t \subset W \cup V_t \cup \{\rightarrow\} \cup \{\ ,\}$

where, $\overline{A}$ is the starting symbol;

W is the universe of nonterminal symbols;

{,} is used to separate the rules of R.

If a grammar-grammar $\overline{G}$ is given, the information required to select G depends on the probability of generating G by $\overline{G}$. The third situation is to define a standard stochastic grammar-grammar $\overline{G}$, that generates grammars on the vocabulary of the given G, and to determine the probability of generating G using this particular $\overline{G}$.

Let G be the context-free grammar whose productions are

$A_1 \rightarrow x_1, \quad A_2 \rightarrow x_2, \ \ldots\ldots \ A_r \rightarrow x_r$

where $x_1, x_2, \ldots, x_r$ are strings in V_t^+, and $A_1, \ldots, A_r$ are elements of V_n. The complexity of G is the sum of the complexities of its productions. Since G is context-free, the complexity of a production is determined by the complexity of its right hand side. Thus $C(G) = \sum_{i=1}^{r} C(x_i)$

We now formulate the computation of $C(y)$, $y \in V_t^+$. Suppose that y has length K and involves the symbols $y_1, \ldots, y_s$, say $k_1, \ldots, k_s$ times, respectively, so that $\sum_{i=1}^{s} k_i = K$. Let "#" be a special "stop symbol". The string y# can be generated by the stochastic grammar G which has the single alternative set

$$A \rightarrow y_1A|\dots|y_sA|\# \qquad (k_1/K+1,\dots,k_s/K+1,1/K+1)$$

The probability of generating y#, i.e., of generating y then stopping, is

$$(1/K+1) \prod_{i=1}^{s} (k_i/(K+1))^{k_i}$$

The complexity of y can be measured by the negative logarithm of this probability.

$$C(y) = \log(K+1) + \sum_{i=1}^{s} k_i \log((K+1)/k_i)$$

$$= (K+1) \log (K+1) - \sum_{i=1}^{s} k_i \log k_i$$

The complexity measure C(y) does not consider the order in which the symbols y_i occur in y. A more complex measure would involve the conditional probabilities of the symbols, derived from their relative conditional frequencies in the order in which they appear in y.

If G is a stochastic context-free grammar and has the productions

$$A_1 \longrightarrow x_{1,1}|x_{1,2}\dots|x_{1,m1} \qquad (P_{1,1},P_{1,2},\dots,P_{1,m1})$$

.

.

$$A_r \longrightarrow x_{r,1}|x_{r,2}\dots|x_{r,mr} \qquad (P_{r,1},P_{r,2},\dots,P_{r,mr})$$

where $A_i = A_j$ for $i = j$, and $\sum_j P_{i,j} = 1$, $1 \leq i \leq r$, then the complexity of G is the sum of complexities of these alternative sets,

$$C(G) = \sum_{i=1}^{r} C(A_i \rightarrow x_{i,1}|\dots|x_{i,mi})$$

The information conveyed by choosing one of the alternatives, say x_{ij}, is $-\log P_{ij}$. Thus

$$C(G) = \sum_{i=1}^{r} \sum_{j=1}^{m} (-\log P_{i,j} + C(x_{i,j}))$$

where the first term represents the information inherent in the jth choice, and the second term represents the complexity of the chosen right hand side, as derived above.

Cook showed that if the length of y (right hand side of a production) increases, while the relative frequencies of the symbols remain the same, then C(y) increases. In addition, if the length of y remains constant, while the number of symbols (other than #) that occur in y increases, then C(y) increases. Finally, Cook showed that for a given length of y and a given number of symbols, C(y) is greatest when all the symbols (except for #) have equal frequencies.

These properties of C(y) are useful when trying to find grammars that are simpler than a given one. These properties will not be of much use for finite-state grammars, since the right hand sides of productions have lengths less than three, and all symbols occurring on the right hand sides of productions are different. However, the complexity measure is not dependent on the right hand sides of productions and their probabilities. The complexity measure is still valid for measuring the complexity of a finite-state grammar. The context-free grammar in Chomsky normal form has the same drawback. However, the second property is useful in searching for a less complex Chomsky normal form grammar.

Several remarks need to be made in connection with this complexity measure. These remarks will be the basis for the heuristic merging procedures in Chap. V.

(1) If a new production shortens and decreases the number of symbols in the original production, then that production decreases the complexity.

(2) Decreasing the length of the right hand side of a production while increasing the number of different symbols increases complexity.

(3) The complexity of a production of the form $X \rightarrow a^n$ cannot be reduced by breaking it down into shorter productions.

(4) If a substring longer than two symbols occurs repeatedly on the right hand side of a production, a decrease in complexity is possible by substituting a symbol for the entire substring.

(5) If the given grammar's productions contain many disjuncts, a reduction in complexity may be achieved by adding a disjunctive production.

4.4. DIFFERENCE MEASURES BETWEEN LANGUAGES

In approximate language identification, the difference between the sample strings and a language can be used to decide the degree of appropriateness of the language in describing the sample. In grammatical inference by the constructive method, it is very important to determine whether the language generated by an inferred grammar can sufficiently represent the sample strings or not. This question will be studied for finite-state grammars and context-free grammars.

4.4.1. GENERAL DISCUSSION

Three different approaches to the notion of difference measure between languages have been introduced in Sec. 2.3.3. We will briefly describe these measures before we define another difference measure. Cook /11/ defined a discrepancy measure for measuring the difference between a probabilistic sample S and a p-language L(G) based on the probability distribution of a language and a set of sample strings, which is formulated as:

$$D(L(G),S) = \sum_{x \in V_t^+} |p(x)(-\log p(x)+C(x))-q(x)(-\log q(x)+C(x))|$$

$$C(x) = (K+1)\log(K+1) + \sum_{j=1}^{s} k_j \log k_j$$

where p(x) and q(x) are the probabilities of x in S and L(G), respectively;

K is the length of x;

s is the number of different symbols in x;

k_j is the number of times the jth symbol occurs in x.

The discrepancy between S and L(G) depends on the lengths, numbers of different symbols, and probabilities of the strings in S and L(G). For infinite languages, there is no guarantee that the discrepancy will converge. The value of the discrepancy ranges from zero to infinity. This discrepancy measure might not show the actual relationship between S and L(G). For instance, if L(G) and S have a complement relationship, the discrepancy can be any value from zero to infinity. Also, S and L(G) are probabilistic languages, but the sum taken over all x in V_t^+ is not consistent with a probabilistic word function. The difference measure of languages will be defined to overcome these disadvantages.

Wharton /67/ defines metrics on the class of languages Z over a finite terminal vocabulary V_t which measure the difference between two strings. Two kinds of metrics are defined: the discrete metric and the weight metric. The discrete metric is used in exact language identification and is not discussed here. The weight metric is based on the sequence of weights which are assigned to words in V_t^+ according to their lexicographic order, and norm. The sum of the weights in a sequence of weights is required to be bounded, and individual weights w_i must be positive. If the sum of weights is equal one then the sequence of weights is normalized. In this case, V_t^+ is a probabilistic language. This difference measure cannot reflect the difference between two probabilistic languages, since the weights do not depend on the original probabilities. Although the weight metric has a several interesting properties, it is not appropriate for this research, because we infer p-grammars.

Maryanski /46/ defined several distance measures for p-languages, based on differences in word probabilities. Some distance measures have bounded values but some do not. The most useful distance measures are the absolute and square difference measures, defined as follows.

Let $p_i(x)$ be the probability of x in the ith language. Then the absolute and square differences for approximating p-language L_1 by L_2 are:

$$D_a(L_1,L_2) = \sum_{x \in L_1} |p_1(x) - p_2(x)|$$

$$D_s(L_1,L_2) = \sum_{x \in L_1} (p_1(x) - p_2(x))^2$$

The distance measures between two p-languages do not involve in the subset L of the p-language L_2 whose strings are not in the p-language L_1. Therefore the distance measures actually do not measure absolute difference between two p-languages; that is $D_a(L_1,L_2) \neq D_a(L_2,L_1)$, making these distance measures inadequate to some extent. If L_1 is finite and the size of L_2 increases monotonically, then the distance between L_1 and L_2 will increase monotonically. This property is not desirable when trying to select the best language to represent L_1.

The difference between two languages L_1 and L_2 can be divided into three parts which are related to three subsets; L_1-L_2, L_2-L_1, and $L_1 \cap L_2$. The subset L_1-L_2 contains strings in L_1 but not in L_2, L_2-L_1 is interpreted analogously; $L_1 \cap L_2$ contains strings in both L_1 and L_2. The difference contained in $L_1 \cap L_2$ is the word probability difference. However, the difference provided by those strings in L_1-L_2 or L_2-L_1 consists of both the word probability difference and string syntactical difference. If L_1 and L_2 contain the same strings then either Cook's discrepancy measure or Maryanski's distance measures may indicate the difference between L_1 and L_2 properly. If there are some strings in L_1 and not in L_2, or vise versa, these two difference measures may not indicate the difference properly.

Among all difference measures for languages, the discrepancy measure based on information theory best meets our intuitive requirements for difference measure. This measure considers both the word probability difference and the string difference.

4.4.2. DEFINITIONS AND PROPERTIES

Several difference measures between two p-languages will be defined in this section, based on Cook's discrepancy measure D(L(G),S) given in Sec. 4.4.1. As mentioned previously, there is no guarantee of convergence for an infinite language. The value of discrepancy ranges from zero to infinity. Since S has a finite cardinality, we could sum only over those x in S, so that the discrepancy would be bounded from above. However, the loss of information would be significant, and the value of discrepancy would not reflect the actual difference between S and L(G). In order to find a measure which will minimize the information loss, and will converge, we will limit the number of strings counted in measuring the discrepancy, as explained below.

For a finite terminal vocabulary V_t, and a positive integer $m<\infty$, there are at most a finite number of strings whose lengths are less than m. This suggests a way to limit the number of strings involved in the calculation. Later in this section, we will use this idea to define a difference measure for languages and discuss its properties.

Another way to limit the number of strings involved in the calculation is to use the probability. Since L(G) is a p-language, the sum of all string probabilities in L(G) equals one. This property can be used in limiting the number of strings in L(G) which should be involved in the calculation of the distance between S and L(G). Suppose we want ninety-five percent of strings in L(G) to be involved in calculating the distance. We simply select strings from L(G) in some prescribed order until the sum of the string probabilities equals 0.95. In calculating the distance between S and L(G) we require that the strings occurring in both S and L(G) should be considered first. Therefore we have to divide L(G) into two subsets L_1 and L_2; L_1 has the same strings as S, L_2 has strings which are in L(G) but not in S. The selection procedure is described as follows:

(1) Set the proportion of strings in L(G) to be involved in the computation, say, q.

(2) Sum the string probabilities in L_1. Suppose the sum is q'.

(3) If $q \leq q'$ then all strings in L_1 are to be used in the computation.

(4) If $q > q'$ then a string in L_2 is selected according to its lexicographic order, and its probability is added to q'.

(5) Continue step (4) until $q' \geq q$. Then the computation involves all strings in L_1 plus those strings selected from L_2.

In computing the distance between S and L(G), we have proposed two different ways to limit the number of strings which should be in the computation. Now we formulate these ideas and define a new distance measure.

In using the length of a string to restrict the number of strings in the computation, let $m = \max_{x \in S} \{\ell(x)\}$, and let $L' = \{x | x \in L(G), \ell(x) \leq m\}$.

In using the string probability to restrict the number of strings in the computation. Let the proportion of L(G) which will be in the computation be q, $0 \leq q < 1$,

let lex(x) denote the lexicographic order of a string x in L(G), and let card(L) denote the cardinal number of a language L. Furthermore, let $L_2 = L(G) - L_1$, where $L_1 = S$, and let $\sum_{x \in L_1} p(x) = q'$. Let $L' = L_1 \cup L_3$, where $L_3 \subset L_2$ and satisfies:

(1) For all $x_i \in L_3$, and all $x_j \in L_2 - L_3$, $\text{lex}(x_i) < \text{lex}(x_j)$.

(2) $\text{Card}(L_3) = \text{Min}_i \{\text{card}(V_i) | V_i \subset L_2, \text{ and } \sum_{x \in V_i} p(x) \geq q - q'\}$.

(3) $\sum_{x \in L_3} p(x) \geq q - q'$

Several distance measures between a p-sample S and a p-language L(G) can now be defined. Let

$$B(x) = p(x)(-\log p(x) + C(x)) - q(x)(-\log q(x) + C(x))$$

where p(x), q(x), C(x) are defined as before.

Definition: The <u>absolute distance</u> between S and L(G) is:

$$D_a(S,L(G)) = \sum_{x\in L'} |B(x)|$$

Lemma 1: The absolute distance between S and L(G) is bounded above.

Proof: Let $m = \max_{x\in S}\{\ell(x)\}$, and let $L' = \{x | x\in L(G), \ell(x) \leq m\}$,

G has terminate set V_t. Since V_t has a finite cardinality, L' has a finite cardinality. We only need to prove that the distance for individual strings is bounded above.

Assume L' has the cardinality n. The absolute distance between S and L(G) is:

$$D_a(S,L(G)) = \sum_{x\in L'} p(x)(-\log_2 p(x) + C(x)) - q(x)(-\log_2 q(x) + C(x))$$

$$\leq n.\max\{|p(x)-q(x)C(x)-p(x)\log_2 p(x) + q(x)\log_2 q(x)|\}$$

$$\leq n.\max\{|p(x)-q(x)C(x)|+|p(x)\log_2 p(x)|+ |q(x)\ \log_2 q(x)|\}$$

$$|p(x)-q(x)C(x)| \leq C(x) = (K+1)\log_2(K+1) + \sum_{j=1}^{s} k_j \log_2 k_j$$

$$<(m+1)\log_2(m+1) + m\cdot(m\log_2 m) < \infty$$

Let $f(x) = x\log_2 x$, then $f(1) = f(0) = 0$.

$f'(x) = 1 + \log_2 x$ when $x = 1/2$, $f'(1/2) = 0$

$f''(x) = 1/x$, $f''(1/2) = 2$

Therefore f(x) has a minimum at x = 1/2.

Thus $|p(x)\log_2 p(x)| \leq 1/2, \quad 1 \geq p_i \geq 0$

$|q(x)\log_2 q(x)| \leq 1/2, \quad 1 \geq q_i \geq 0$

$$D_a(S,L(G)) < n((m+1)\log_2(m+1) + m^2\log_2 m+1) < \infty$$

Let $n((m+1)\log_2(m+1) + m^2\log_2 m + 1) = N$.

Then $D_a(S,L(G))<N<\infty$ is bounded above.

Let q be the portion of L(G), $0\leq q<1$, and let

$L' = L_1 \cup L_3$, where $L_1 = S$, and L_3 is defined as before. Since S is a finite set, L_1 has a finite cardinarity. We want to prove that L_3 has a finite cardinality.

Let $\sum_{x\in L'} p(x) = q_1$. If $q_1 \geq q$, then $L_3 = \emptyset$, so L_3 has zero cardinality.

If $q_1<q$, then a string in L_2 will be selected according to its lexicographic order, and its probability will be added to q_1. If an infinite number of strings is selected from L_2, then all strings in L_2 will be selected because of the lexicographic order. Thus $q_1 = q = 1$ which violates our assumptions - Q,E,D.

Definition : The square difference between S and L(G) is:

$$D_s(S,L(G)) = \sum_{x\in L'} (B(x))^2$$

Lemma 2: The square distance is bounded above.

Proof: Analogous to the proof of Lemma 1.

$$D_s(S,L(G)) \leq n\cdot N^2<\infty$$

Definition : The mean absolute difference between S and L(G) is:

$$D_{ma}(S,L(G)) = \sum_{x\in L'} |B(x)|q_i$$

Lemma 3: The mean absolute distance is bounded above.

Proof: Analogous to the proof of Lemma 1.

Since $0 \leq q(x) \leq 1$, therefore $D_{ma}(S,L(G)) < n \cdot N < \infty$

<u>Definition</u> : The mean square distance between S and L(G) is:

$$D_{ms}(S,L(G)) = \sum_{x \in L'} (B(x))^2 q_i$$

Lemma 4: The mean square distance is bounded above.

Proof: Analogous to the proof of Lemma 1.

Since $0 \leq q(x) \leq 1$, therefore $D_{ms}(S,L(G)) \leq n \cdot N^2 < \infty$

We are only interested in distance measures having an upper bound so that we will not discuss the relative distance between two p-languages. These distance measures between S and L(G) have the symmetric and the transitive properties. The distance measures can be used as an acceptance criteria for inferred grammars.

<u>4.5</u>. <u>STATISTICAL ACCEPTANCE CRITERION OF LANGUAGES</u>

In approximate language identification, a grammar is selected whose language strongly approximates the sample /67/. In grammatical inference by constructive methods, the acceptability of a grammar is based on the difference between its language and the sample. A discrepancy measure based on information theory measuring the difference between a p-language and a p-sample and a number of distance measures between p-languages has been developed in Sec. 4.4. However, the actual threshold value of a difference measure for an acceptance decision must depend on subjective ideas and inference procedures. The Bayes' decision rule and the chi-square test for distribution difference have been adopted /35/,/46/ for this purpose.

The difference measures between languages defined in Sec. 4.4. are applied to grammatical inference in this section. The Bayes' decision rule will not be discussed here because the conditions for applying this method are not compatible with the problem under study. The chi-square test for distributional difference and probability independence will be the most important tests in this thesis.

In this section, we will discuss statistical tests and acceptance criteria based on a language and a sample. Some distribution-free statistical tests will be presented and a comparison will be made between them and the chi-square test. The testing hypothesis and the characteristics of tests will also be discussed.

4.5.1. APPROPRIATENESS OF A LANGUAGE FOR A SAMPLE

In using statistical procedures and tests to decide whether a sample can be appropriately represented by a language, the parameters used in the statistics must be carefully considered. Deciding what should be measured to demonstrate the significance of a language in representing a sample is very difficult. In this section, we will discuss the problem of selecting parameters for statistical procedures.

The finite sample S may be characterized by the following features:

(1) The set of different symbols in S;

(2) The maximum lengths of strings in S;

(3) The length distribution of strings in S;

(4) The symbol distribution for strings of different lengths;

(5) The frequency distribution of strings in S;

(6) The dissimilarity matrix of strings in S.

The appropriateness of a language for representing a sample might be decided by these features. If the frequency distribution of strings is not uniform, then the sample is a p-sample and will be discussed later in this section. The dissimilarity matrix for a set of strings does not depend on the language to which the strings belong. Therefore, it cannot be used in comparisons. If the maximum lengths of strings allowed are specified, the language has to be a finite language. The length distribution and symbol distribution of lengths may be used in deciding the appropriateness of a language for a sample. One must also consider whether an infinite language can reflect the degree of infiniteness inherent in the sample. There is no general way to measure the degree of infiniteness.

The fact that no obvious techniques exist for measuring the appropriateness of a language for a sample does not mean that such a measure is not necessary. Rather, it implies that finding an appropriate parameter for comparison is very difficult.

Most studies concentrate on the appropriateness of a p-language for a p-sample since the probability distribution of strings can be used in statistical measures. The probability of strings in the sample can be estimated from the string frequency distribution. In this thesis, a p-grammar is inferred for a p-sample. The techniques based on the probability measure will be most useful in deciding the degree of appropriateness of a p-language for a p-sample.

4.5.2. ACCEPTANCE CRITERION BASED ON STATISTICAL TESTS

In grammatical inference, the chi-square goodness of fit test has been used to test the difference in string frequencies between a p-sample and a p-language (Cf. Sec. 2.3.2.). In this section, the chi-square test will be investigated as a statistical acceptance criterion. The distribution-free Kolmogrov-Smirnov maximum deviation test will be presented and compared to the chi-square test.

4.5.2.1. CHI-SQUARE GOODNESS-OF-FIT TEST

In this section, a brief description of the chi-square test mentioned in Sec. 2.3.2. will be given and used to measure the difference between observed and expected frequency distribution. The use of the chi-square test in grammatical inference will also be investigated.

Let $F(\cdot)$ be some completely specified, hypothesized distribution function, and let p_i be the probability of a random observation being in category i, $i=1,\dots,k$, $\sum_{i=1}^{k} p_i=1$. If a random sample of size n is taken, with o_i being the number of observations in category i, $\sum_{i=1}^{k} o_i = n$, then the expected number of observations in category i is $e_i = np_i$. The joint distribution of o_1, $o_2 \ldots o_k$ is multinomial.

$$Pr(o_1,o_2,\ldots o_k) = (n!/o_1!o_2!\ldots o_k!)\,p_1^{o_1}\,p_2^{o_2}\cdots p_k^{o_k}$$

By Stirling's approximation,

$$Pr(o_1,o_2,\ldots o_k) = C \exp\,(-1/2(\sum_{i=1}^{n}\,(o_i-np_i)^2/np_i))$$

where C is a constant for given values of n, k and the various p_i. If n ∞, then the following statistic has an approximate chi-square distribution with (k-1) degree of freedom.

$$\chi^2_{k-1} = \sum_{i=1}^{k}(o_i - np_i)^2/np_i$$

The testing hypothesis and acceptance criterion had been stated in Sec. 2.3.2. The table for χ^2- test can be found in Lindgren /44/.

Let $E = \{(x_1,c_1)\ldots,(x_k,c_k)\}$ be an experimental set, where x_i is a string and c_i is the corresponding frequency. Each x_i can be treated as a category. Thus E has k categories. Let $\sum_{i=1}^{k} c_i=m$. Then the empirical probability of an observation (string) being in the category i is $p_i = c_i/m$. Let L be a p-language and for each $x_i \in L$, $i=1,2,\ldots k$, $p'(x_i)=p_i'$ is the corresponding probability of x_i in L. In applying the chi-square statistic to grammatical inference, the difference between the expected frequency f_i and the observed frequency mp_i' is examined. The chi-square statistic becomes:

$$\chi^2_{k-1} = \sum_{i=1}^{k}(o_i-e_i)^2/e_i = \sum_{l=1}^{k}(mp_l' - c_i)^2/c_i$$

$$= \sum_{i=1}^{k} (mp_i' - mp_i)^2/mp_i = \sum_{i=1}^{k} m(p_i' - p_i)^2/p_i$$

Chi-square tests are inexact unless the assumption of infinite observed frequency in each category and the minimum expected frequency in each category are met. The following hazards are associated with chi-square tests.

(1) If some of the expected frequencies are small, the asymptotic chi-square distribution may not be appropriate.

(2) The sample size must be large.

(3) The prohibition against small expected frequencies has led to the widely accepted practice of pooling categories in order to bring the expected frequencies for the combined categories up to the required size. Such pooling, however, involves an arbitrary decision which changes the character of the test (i.e., of the test hypothesis). Also it violates the assumption of random sampling.

The above discussion shows that the chi-square test is not entirely suited to grammatical inference. The observed sample is not randomly drawn from a p-language, and the sample size cannot be infinite. Also the string probability is estimated from the experimental set which may not reflect the exact string probability, since an experimental set contains fewer categories than a language.

4.5.2.2. THE KOLMOGROV-SMIRNOV MAXIMUM DEVIATION TESTS OF AN HYPOTHESIZED POPULATION DISTRIBUTION

The Kolmogrov-Smirnov (K-S) tests are appropriate for testing the hypothesis that two samples are from the same distribution and are the best known of many "maximum deviation" tests. In this section, we will discuss the K-S tests and their use in grammatical inference.

The assumptions of the test are:

(1) Sampling is random, and there are no tied observations.

(2) The sample distribution is continuously distributed.

(3) The hypothesized distribution must be specified completely, without regard to any information contained in the sample.

For finite sample size, the test is biased /47/. The test, however, can be applied to discrete populations /40/. The K-S tests used in testing the difference between a p-language and a p-sample is described below.

Let E, x_i, c_i, be defined as before, and let $F_n(\cdot)$ be the cumulative probability distribution of x, a random variable, which has k values $x_1, \ldots x_k$. Then

$$F_n(x) = F(x \leq x_n) = \sum_{i=1}^{n} p(x_i).$$

Let L be a p-language, and for each $x_i \in L$, let $p'(x_i)$ be the corresponding probability. Let $S_n(\cdot)$ be the empirical cumulative probability distribution of the n obtained observations on X. Thus

$$S_n(x) = S(x \leq x_n) = \sum_{i=1}^{n} p'(x_i).$$

Then the K-S maximum deviation statistics are defined as:

$$K = \max_x \left[S_n(x) - F_n(x) \right]$$

$$K^{+} = \max_x \left| S_n(x) - F_n(x) \right|$$

$K^- = \min_x |S_n(x) - F_n(x)|$

Let F_0 be the true cumulative distribution function of the sampled variate. Then the testing hypotheses are:

$H_0 : F_n(x) = F_0(x)$ for all x

$H_1 : F_n(x) = F_0(x)$ for some x; use K statistic

$H_1 : F_n(x) > F_0(x)$ for some x; use K^+ statistic

$H_1 : F_n(x) < F_0(x)$ for some x; use K^- statistic

Birnbaum /4/ has provided tables to five decimal places. Accept H_0 at significance levels α, if K is less than K' in the table for the given n and α.

For the one-sided test K^+, accept H_0 at significance level α, if K^+ is less than K' found in the table for the given n and α. For K^- statistic, accept H_0 at significance level α, if K^- is greater than K' found in the table for the given n and α.

Since the sensitivity of K is not concentrated upon a particular type or class of alternatives, it is, in effect, a test of goodness of fit. The most appropriate classical test against which to compare it is the chi-square test. The K-S test is superior to chi-square tests in the following ways.

(1) The K-S test requires only the relative modest assumptions that sampling is random and the sample population is continuous, whereas the chi-square test assumes that the sample size is infinite.

(2) The exact distribution of K is known and tabled for small sample size, whereas the exact distribution of the chi-square test is known and tabled only for infinite sample size.

(3) The chi-square test is only an approximate test, at all sample sizes, and the degree of approximation is difficult to assess, whereas the K-S test is exact at small sample sizes and its degree of approximation at large sample sizes is more readily assessable.

(4) The K^+ and K^- test statistics were designed to test for deviations in a given direction, and do so easily, whereas the chi-square test must be specially modified and conducted in unconventional fashion in order to do so.

(5) The K-S test uses ungrouped data, whereas the chi-square test uses grouped data.

The chi-square test on the other hand, is superior to the K-S test in the following ways:

(1) The chi-square test does not require that the hypothesized population be completely specified in advance of sampling.

(2) The chi-square test can be applied to discrete populations, but not the K-S test. When the assumption of continuity is not met, the K-S test is conservative.

(3) The chi-square values can be meaningfully added by making the appropriate reduction in degree of freedom.

Since all the assumptions for using K-S test are not usually met in grammatical inference, the K-S test, just like the chi-square test, is not an optimum test for grammatical inference. The major difficult lies in the random experiment. In general, we assume that the experiment set is a random sample. But since an infinite language contains an infinite number of strings and the actual string distribution is unknown, it is technically impossible to decide the sample size for obtaining an unbiased estimate of string probability.

Although both chi-square and K-S tests have many disadvantages in grammatical inference, they still are valuable tests for deciding which language is most appropriate for a sample. Presumably, the inaccuracy caused by unsatisfied conditions on the test is the same from language to language. Since the conditions under which the test is applied are the same for all candidate languages of a sample, a constant error is expected in the tests.

4.6. SUMMARY

This chapter concentrates on the definitions of solution grammar, complexity measures and statistical acceptance criteria for grammatical inference. The problem of defining solution grammars for grammatical inference by constructive methods has been discussed. The solution grammar for this thesis is defined and characterized by grammatical complexity and acceptance criteria.

The concept of grammatical complexity measures is reviewed and discussed. A size measure of grammars is defined for evaluating the performance of dissimilarity measures.

A complexity measure based on information theory is defined which will be used to optimize the selection of grammars.

Difference measures between p-languages have been reviewed and discussed. Several difference measures based on information theory are defined and their properties are investigated. The proposed difference measures are shown to be more realistic than those in the literature.

The use of the chi-square goodness of fit test and its application to grammatical inference have been discussed. The Kolmogrov-Smirnov maximum deviation test is proposed and compared to the chi-square test, its application to grammatical inference is discussed.

Techniques presented in this chapter will be used to develop constructive methods in Chap. V.

CHAPTER V

INFERENCE OF PROBABILISTIC GRAMMARS

5.1. INTRODUCTION

In the previous chapters a number of constructive methods for grammatical inference have been discussed. In this chapter, we will integrate these ideas into a new technique for grammatical inference with finite-state and Chomsky normal form p-grammars.

The essential features of any constructive method are:

(1) Grammars are constructed based on a sample of strings and heuristic procedures;

(2) Each constructed grammar is examined and a relatively better grammar is selected;

(3) The language generated by each constructed grammar is compared with the sample, and an acceptance criterion for the constructed grammar is evaluated, based on the difference between the language and the sample.

The problem studied in this chapter is to infer a p-grammar for a p-sample by a constructive method. The p-sample is assumed randomly drawn from a p-language. The type of grammar being constructed is the same as the type of p-language assumed for the p-sample. The constructive procedure for this problem consists of the following components:

(1) Analyzing the syntactical structure of the sample.
(2) Constructing the initial grammar from the sample strings.
(3) Generating candidate grammars from the initial grammars.
(4) Evaluating an acceptance criterion.

Methods for analyzing the syntactical structure of sample strings were presented in Chap. III. Techniques for generating candidate grammars were discussed in Chap. II and IV. Methods for acceptance testing of inferred grammars were described in Chap. IV.

The overall picture of the constructive method proposed in this thesis is described below.

(1) Cluster the sample strings to establish their syntactic structure in an inference tree.
(2) Construct an initial grammar for each cluster by merging the partial grammar for each string, according to the subtree defining the cluster. Then generate candidate grammars for the clusters.
(3) Merge the candidate grammars for the clusters according to the inference tree to produce a candidate grammar for the sample.
(4) Compare the language generated by the candidate grammar and the p-sample. If the language is not acceptable, alter either the syntactic description of structure or merging rules and repeat.

The problem of assigning production probabilities when inferring a p-grammar for a p-sample will be discussed in Sec. 5.2. The procedure for generating candidate p-grammars is divided into two problems. The first problem is to generate an initial p-grammar from the p-sample, based on the inference tree (Step (2)). The second problem is to apply heuristic strategies to merge productions in the initial grammar and construct new p-grammars called candidate p-grammars for the p-sample (Step (3)). These two problems will be discussed in Sec. 5.3. and 5.4. for finite-state and context free grammars.

5.2. ASSIGNING PRODUCTION PROBABILITIES

In this section, we discuss the procedure of assigning production probabilities. The techniques will be used in Sec. 5.3. and 5.4. There is no theoretically-based technique for assigning probabilities to the productions of an ambiguous grammar. Since the problem of assigning production probabilities is secondary in this thesis, the production probabilities will be assigned in a simple way and differently for different situations.

In assigning production probabilities, only the following four situations will be considered. These conditions will apply later to both initial and candidate p-grammars. A is the initial symbol.

(1) Let "ab" be a string with probability P, and let a grammar generating the string contain the productions:

$A \rightarrow aB, \quad B \rightarrow b$

where A, B are nonterminals and a, b are terminals.

The first production in the derivation sequence of the string is assigned the string probability. In this case the production $A \rightarrow aB$ will have probability P and $B \rightarrow b$, probability one.

(2) Let $A \rightarrow aB$, $A \rightarrow aC$ be two productions with probabilities P_1 and P_2 respectively. Let these two productions be replaced by the productions $A \rightarrow aD$, $D \rightarrow B|C$. Then $Pr(A \rightarrow aD) = P_1 + P_2$

Since $A \rightarrow aB$ is replaced by $A \rightarrow aD$ and $D \rightarrow B$. Similarly,

$$Pr(A \rightarrow aD)\ Pr(D \rightarrow B) = P_1$$
$$Pr(A \rightarrow aD)\ Pr(D \rightarrow C) = P_2$$

Thus,

$$Pr(D \rightarrow B) = P_1/(P_1 + P_2)$$
$$Pr(D \rightarrow C) = P_2/(P_1 + P_2)$$

(3) Let two productions and their probabilities be

$A \rightarrow B_3B_1|B_2B_2 \quad (P_{11}, P_{12}), \quad C \rightarrow B_3B_1|B_2B_2 \quad (P_{21}, P_{22})$

If A and C are replaced by a new nonterminal U, then the probability of the new production is assigned as the average of the old production probabilities

$$Pr(U \rightarrow B_3B_1|B_2B_2) = ((P_{11} + P_{21})/2,\ (P_{12} + P_{22})/2)$$

(4) Let a subset of the productions for a grammar and their probabilities be

a) $A \rightarrow cB_1|dD_1 \quad (P_{11}, P_{12})$, b) $A \rightarrow cB_2|dD_2 \quad (P_{21}, P_{22})$,

c) $B_1 \rightarrow b|c \quad (P_{31}, P_{32})$, d) $B_2 \rightarrow aB_1|a \quad (P_{41}, P_{42})$,

e) $D_1 \rightarrow aD_2$, f) $D_2 \rightarrow e$

If B_1 and B_2 are replaced by a new nonterminal B, then the productions a) and b) are merged into

$A \longrightarrow cB|dD_1|dD_2 \quad (P_{11} + P_{21}, P_{12}, P_{22})$.

Since the productions c) and d) are related to a) and b), a new production

$B \longrightarrow b|c|aB|a$ is created. The production probabilities are assigned as follows:

$$\Pr(B \longrightarrow b) = P_{11} \times P_{31}/(P_{11} + P_{21})$$
$$\Pr(B \longrightarrow c) = P_{11} \times P_{32}/(P_{11} + P_{21})$$
$$\Pr(B \longrightarrow aB) = P_{21} \times P_{41}/(P_{11} + P_{21})$$
$$\Pr(B \longrightarrow a) = P_{21} \times P_{42}/(P_{11} + P_{21})$$

These four conditions include all cases occurring in the inference procedure in subsequent sections.

5.3. INFERENCE OF FINITE-STATE P-GRAMMARS

According to Sec. 5.1. any constructive procedure usually consists of three subproblems: analyzing the syntactical structure of a p-sample, constructing candidate p-grammars and acceptance testing of inferred p-grammars. The first and the third subproblems have been carefully discussed in Chap. III and IV. The subproblem of constructing candidate finite-state p-grammars (FSPG) will be discussed in this section. This subproblem is divided into two steps: (1) constructing initial finite-state p-grammar, and (2) heuristically merging productions to produce candidate p-grammars. These two steps will be described separately. The procedures described below are particularly effective with the inference tree produced in Sec. 3.4.

5.3.1. CONSTRUCTING INITIAL FINITE-STATE P-GRAMMAR

The initial finite-state p-grammar exactly generates the p-sample and is constructed based on the results of the cluster analysis of sample strings. The following preliminary definitions are needed:

Definition: A canonical finite-state p-grammar (CFSPG) $G_c = (V_n, V_t, R, P, A)$ for a string $x = a_1 \ldots a_n$ with probability p is defined below.

$V_n = \{A, A_1, \ldots, A_{n-1}\}$ is a set of nonterminals;

$V_t = \{a_1, a_2, \ldots\ldots, a_n\}$ is a set of terminals;

A is the start symbol;

R is the set of productions, defined recursively as

$A \rightarrow a_1A_1$, $A_{i-1} \rightarrow a_iA_i$ for $2 \leq i \leq n-1$, $A_{n-1} \rightarrow a_n$

P is the set of production probabilities. All productions have probability one except for the production $A \rightarrow a_1A_1$, which has probability p.

Definition: Two productions are equivalent if and only if they have the same probabilities and identical right hand sides.

The procedure for constructing the initial finite-state p-grammar from a p-sample consists of two steps. First, construct a CFSPG for each cluster or subtree of the labelled MST (Sec. 3.5.). Then combine all CFSPG and make necessary consolidations. For each cluster, the construction starts at the shortest string in the cluster, then follows the depth first order in the labelled MST which considers top to down first, left to right second, to construct a CFSPG for each string in the tree. After all strings in a cluster are visited, productions are grouped together to form an initial p-grammar for the cluster.

Since the initial finite-state p-grammar has to generate the sample strings, the consolidation procedure is limited to replacing equivalent productions without creating recursive productions. When two productions are equivalent, one of them is eliminated, and the occurrences of the left hand symbol of the eliminated production are replaced by the left hand symbol of the other production.

The following example demonstrates the procedure of constructing the initial finite-state p-grammar.

Example 1: Let S_1 = {cb, cab, caab, caaab } be a set of strings with corresponding probabilities {1/4, 1/8, 1/16, 1/16}. The labelled MST connects the strings in the sequence shown with root cb, string cab at depth 1, string caab at depth 2 and string caaab at depth 3.

The stepwise construction is as follows; where capital letters are nonterminals, unless otherwise stated, all productions have probability one.

(1) Construct the CFSPG for the string "cb"

$A \rightarrow cB_1$ (1/4), $B_1 \rightarrow b$.

(2) Construct the CFSPG for the second string "cab"

$A \rightarrow cB_2$ (1/8), $B_2 \rightarrow aB_3$, $B_3 \rightarrow b$

(3) The productions $B_1 \rightarrow b$ and $B_3 \rightarrow b$ are equivalent. Delete $B_3 \rightarrow b$ and change all occurrences of B_3 to B_1. Thus, we obtain:

$A \rightarrow cB_1$ (1/4), $B_1 \rightarrow b$,$A \rightarrow cB_2$ (1/8), $B_2 \rightarrow aB_1$.

Repeating this procedure, we obtain the initial finite-state p-grammar G_1 for S_1, which has the following set of productions.

$A \rightarrow cB_1$ (1/4), $B_1 \rightarrow b$, $A \rightarrow cB_2$ (1/8), $B_2 \rightarrow aB_1$, $A \rightarrow cB_3$ (1/16), $B_3 \rightarrow aB_2$, $A \rightarrow cB_4$ (1/16), $B_4 \rightarrow aB_3$.

5.3.2. CONSTRUCTING THE CANDIDATE FSPG

In this section, we propose a procedure for constructing candidate finite-state p-grammars from an initial finite-state p-grammar. Candidate finite-state p-grammars are constructed to reduce the complexity of the initial finite-state p-grammar. According to the grammatical complexity measure defined in Sec. 4.4., the complexity of a grammar can be reduced by introducing recursive and disjunctive productions. A recursive production is a production which has the same symbol occurring on both sides of the production. A disjunctive production is one with more than one branch on the right hand side of the production. The heuristic constructive procedure consists of the following steps and is discussed in this section.

(a) Creating recursive and disjunctive productions by merging related productions;

(b) Consolidating the new grammar and removing undesirable productions which contribute unnecessary complexity.

5.3.2.1. RULES FOR MERGING PRODUCTIONS AND CONSOLIDATION

In this section, we discuss the merger of productions to form recursive or/and disjunctive productions. In setting up merger rules, we will consider the relationships among symbols occurring in different productions. Since the length of the right hand side of each production in any finite-state grammar cannot exceed two, the symbol relationships between two productions are limited. All possible situations and corresponding consolidations are listed below. When two productions are to be merged, the rules are applied in the order shown (see Appendix B).

(1) Eliminating one of the productions.

When two productions are equivalent, for example $A \rightarrow aB$ and $C \rightarrow aB$, one of them is eliminated and the occurrences of the left hand symbol of the eliminated production are replaced by the left hand symbol of the other production.

(2) Introducing a recursive production.

When the terminals and nonterminals are related as in $A \rightarrow aB$, $B \rightarrow aA$, a recursive production $A \rightarrow aA$ is created.

(3) Introducing a recursive and disjunctive production.

The situation is similar to (2), as in $A \rightarrow aB$, $B \rightarrow aC$. The production $A \rightarrow aA|aC$ is created.

(4) Introducing a disjunctive production.

When two productions have identical left hand symbols, for example, $A \rightarrow aB$, $A \rightarrow bB$ they are merged into $A \rightarrow aB|bB$.

(5) Introducing a new nonterminal.

When two productions only differ in the right hand nonterminal, for example $A \rightarrow aB$, $A \rightarrow aX$, a new nonterminal is introduced to replace all occurrences of B and X.

(6) Creating recursive productions.

When the situation $A \rightarrow aB$, $B \rightarrow bC$ arises, productions with C as the left hand symbol or A as the right hand symbol are considered. If applying the sequence of productions shows that the terminal symbols occur repeatedly with the same pattern, recursive productions may be created to replace the set of productions.

5.3.2.2. SEARCH STRATEGY

The rules in Sec. 5.3.2.1. show how pair of productions are merged. In this subsection, strategies for selecting pairs of productions for merging from those in the initial finite state p-grammar are proposed.

The search strategy uses the labelled MST to investigate the possibility of merging productions. The search process will use a depth first search to discover recursive productions as well as disjunctive productions. The constructive procedure first tries to merge productions within a cluster or subtree. According to the depth first order, the productions generating a string are examined along with those productions generating its predecessors for merging productions. Productions from all clusters are then examined according to the inference tree for further consolidation. The procedures for merging productions from different clusters are similar to those for merging productions in a cluster.

Several important heuristic tactics are considered.

(1) Productions with the initial symbol at the left hand are not allowed to merge with other productions except the productions having the same left hand symbol, or when the merging process is at the final stage.

(2) If the set of strings generated by a set of productions is contained in the set of strings generated by another set of productions, then the first set of productions can be eliminated. This should be done whenever a merging is achieved.

(3) When a new nonterminal is introduced to replace a pair of symbols, the appropriateness of replacing each occurrence should be tested (Sec. 4.5.).

<u>Example 2</u>: Let sample S_1 and initial finite-state p-grammar G_1 be defined as in Ex. 1. The merging process starts with the productions that generate the

root and its immediate descendant, $A \rightarrow cB_1$ and $A \rightarrow cB_2$. According to Rule 5 in Sec. 5.3.2.1., B_1 is replaced by B_2. Thus, we have $A \rightarrow cB_2$ (3/8), and by Rule 4, $B_2 \rightarrow b|aB_2$ (2/3,1/3) is created. Similarly, merging $A \rightarrow cB_3$ with $A \rightarrow cB_2$, we have $A \rightarrow cB_3$ (7/16), $B_3 \rightarrow b|aB_3$ (4/7, 3/7). Finally, $A \rightarrow cB_4$ and $A \rightarrow cB_3$ are treated the same way, and the final set of productions is:

$A \rightarrow cB_4$ (1/2), $B_4 \rightarrow b|aB_4$ (1/2, 1/2)

<u>Example 3</u>: Let S $\{S_1,S_2\}$ be a p-sample, where S_1 is defined as in Ex. 1, and S_2 = {ab, abb, abbb, abbbb } with associated probabilities {1/4, 1/8, 1/16, 1/16}. By a merging process similar to that in Ex. 2, the final finite-state p-grammar for S_2 has the productions:

$A \rightarrow aC_4$ (1/2), $C_4 \rightarrow b|bC_4$ (1/2, 1/2).

Now we merge productions from the two p-grammars in an heuristic manner. By introducing a new nonterminal B to replace B_4 and C_4, we obtain the productions: $A \rightarrow aB|cB$ (1/2, 1/2)

$B \rightarrow b|aB|bB$ (1/2, 1/4, 1/4).

If the language generated by this grammar is acceptable, then it is the solution grammar. Otherwise, different heuristics must be applied or the syntactic structure must be evaluated.

5.4. INFERENCE OF CONTEXT-FREE P-GRAMMARS

Since every context-free language can be generated by a Chomsky normal form grammar, in the inference of context-free p-grammars, we propose a constructive method for constructing Chomsky normal form p-grammars (CNFPG) for a p-sample. The constructive procedure consists of two steps: first, construct the initial CNFPG from the p-sample, then merge productions to generate candidate p-grammars for the p-sample. These steps will be discussed separately in this section.

5.4.1. CONSTRUCTING THE INITIAL CNFPG FROM A P-SAMPLE

The steps for constructing the initial CNFPG from a set of sample strings are similar to those for constructing the initial finite p-grammar. First, construct a deterministic CNFPG for each cluster by merging CNFPG's for the strings in the cluster, then combine them and make necessary consolidations.

The construction of a CNFPG for a string consists of the five steps described below.

(1) Construct a complete binary tree (CBT) or partial complete binary tree (PCBT) for each string in the cluster based on the length of the string. Let n be the length of the string x, and let k, m be positive integers. Then

(1a) If $n = 2^k$, construct a CBT with k levels;

(1b) If $n = 2^k + m$, where $1 \leq m < 2^{k-1}$ construct a PCBT with k+1 levels in which the left most m nodes at the kth level have two descendents.

(2) Assign a nonterminal symbol to each node in the tree.

(3) Add an edge to each terminal node and assign the symbols in the given string to the added nodes on one-to-one basis from left to right. If the symbol in the given string is a nonterminal, do not add an edge but assign the nonterminal directly to the node in the CBT (or PCBT).

(4) Formalize productions from the tree.
A node and its successors generate a production by using the nonterminal corresponding to the node as the left hand side and nonterminals or terminal corresponding to its successors as the right hand side of the production.

(5) Replace equivalent productions.

In the construction of a CNFPG for a cluster of strings, we exploit the self-embedding property of some context-free languages. The common substring relation between strings in the cluster is used in the construction. The constructive procedure for each cluster consists of the following steps:

(1) Construct a CNFPG for the shortest string in the cluster.

(2) Perform a depth first search in the labelled MST for successors. If no successor nodes exist then stop, otherwise continue.

(3) Substitute substrings in the successor for the existing nonterminals whose sentential forms are equal to the substrings. The substitution takes the longest substring first, and if there are several alternatives then the

left most substring has priority.

(4) Construct a CNFPG for the string in Step 3, and consolidate it with existing productions. Then go to Step 2.

The initial CNFPG for the p-sample is obtained by combining all CNFPG's for clusters and making all necessary final consolidations.

In order to avoid the confusion caused by the use of the initial symbol in different places, we will use different initial symbols for different strings in the cluster. These differences will be eliminated during mergers.

<u>Example 4</u>: Let S_1 be defined as in Ex. 1.

The construction of the initial CNFPG from S_1 is shown below.

(1) Construct a CNFPG for the shortest string 'cb'.

$A_1 \longrightarrow B_1B_2(1/4)$, $B_1 \rightarrow c$, $B_2 \longrightarrow b$.

(2) The descendant of 'cb' is 'cab'. After replacing all substrings in 'cab' by previously-defined nonterminals,'cab' becomes 'B_1aB_2', with probability (1).

(3) Construct a CNFPG for 'B_1aB_2'.

$A_2 \longrightarrow B_3B_2$ (1/8), $B_3 \longrightarrow B_1B_4$, $B_4 \longrightarrow a$.

(4) The descendant of 'cab' is 'caab'. After substituting for all substrings as explained above, 'caab' becomes $B_3B_4B_2$ with probability (1/16).

(5) Construct a CNFPG for '$B_3B_4B_2$'.

$A_3 \rightarrow B_5B_2$ (1/16), $B_5 \rightarrow B_3B_4$.

Repeating the procedure for the remaining strings and renaming nonterminals produces the initial CNFPG, G_1 containing the follow up productions:

$A \rightarrow B_1B_2$ (1/4), $B_1 \rightarrow c$, $B_2 \rightarrow b$, $A \rightarrow B_3B_2$ (1/8), $B_3 \rightarrow B_1B_4$, $B_4 \rightarrow a$, $A \rightarrow B_5B_2$ (1/16), $B_5 \rightarrow B_3B_4$, $A \rightarrow B_6B_2$ (1/16), $B_6 \rightarrow B_5B_4$.

5.4.2. CONSTRUCTING THE CANDIDATE CNFPG

In this section, we will present a procedure for constructing candidate CNFPG's from the initial CNFPG. The strategy for constructing candidate CNFPG is the same as that for candidate finite-state p-grammars described in Sec. 5.3.2.

Optimizing the grammatical complexity measure defined in Sec. 4.4. requires the constructive procedure to merge related productions into disjunctive and/or recursive productions. A procedure for removing undesirable productions which introduce unnecessary complexity is also needed. These two steps will be described in the next two sections.

5.4.2.1. RULES FOR MERGING PRODUCTIONS AND CONSOLIDATIONS

In this section, all possible situations in which two productions from CNFPG's can be merged into disjunctive and/or recursive productions will be discussed. The consolidation for each situation also will be described. The rules are analogous to those for finite-state grammars. The strategy for using these rules is explained in Sec. 5.4.2.2. A complete listing of all situations is given in Appendix B.

(1) Eliminating one of the productions.

When two productions are equivalent, one of them will be eliminated. The necessary consolidation is to replace all occurrences of the left hand nonterminal of the eliminated production by the left hand nonterminal of the remaining production.

(2) Introducing a recursive production.

Suppose two productions meet all the following conditions.

(2a) Both productions have three different symbols.

(2b) Both productions have an identical symbol at the same right hand side position.

(2c) The left hand symbol of a production equals the non-identical right hand symbol of the other production, and vice versa.

Then all occurrences of the left hand symbol of one production are replaced by the left hand symbol of the other production and the second production is eliminated. For example, for $A \rightarrow BC$, $C \rightarrow BA$, the production $A \rightarrow BA$ is created.

(3) Introducing a recursive and a disjunction production.

Suppose two productions meet one of the following conditions.

(3a) The conditions (2a) and (2b) plus the left hand symbol of a production equals the non-identical right hand symbol of the other production, or vice versa.

(3b) A production has all symbols different and the other production has identical right hand symbols. In addition the left hand symbol of a production equals a right hand symbol of the other production, or vice versa.

Then the consolidation is the same as the situation (2), except no production is eliminated. For example, for $A \rightarrow BC$, $D \rightarrow BA$, the production $A \rightarrow BC|BA$ is created.

(4) Introducing a new nonterminal.

Suppose two productions meet all the following conditions.

(4a) The left hand symbols of both productions are identical.

(4b) One of the right hand symbols of both productions are identical at the same position.

(4c) Neither of the non-identical symbols on the right hand sides is identical to the left hand symbol.

Then a new nonterminal is introduced to replace the pair of distinct symbols. For example, for $A \rightarrow BC$, $A \rightarrow BD$, a new nonterminal E is introduced to replace all occurrences of C and D.

(5) Introducing a disjunctive production.

Suppose two productions satisfy both the following conditions.

(5a) The left hand symbols of both productions are identical.

(5b) The condition of (4b) is not satisfied. Then a disjunctive production is generated. For example, for $A \rightarrow BC$, $A \rightarrow CD$, a disjunctive production $A \rightarrow BC|CD$ is generated.

(6) Creating recursive productions.

Suppose two productions satisfy both the following conditions.

(6a) Both productions have three different nonterminals.

(6b) The left hand symbol of one product is identical to one of the right hand symbols, say the first symbol, of the other production. Then we try to consolidate a set of existing productions having all the following properties into recursive productions.

(Q1) All symbols in the production are different.

(Q2) Either the left hand symbol of the production is identical to the first right hand symbol of one production, or the first right hand symbol of the production is identical to the left hand symbol of the other.

If a repeating pattern of nonterminals occurs when the productions are applied, then introduce a recursive production. Otherwise, continue looking for related productions until all productions are examined. For example, for $A \rightarrow BC$, $B \rightarrow DE$, we look for productions which either have A as the first right hand symbol, or B as the left hand symbol, and satisfy Q_1.

In the inference process, these rules are applied in the order they are listed. All relationships between two productions are tabled in the Appendix according to these situations.

5.4.2.2. SEARCH STRATEGY

Since strings having a close syntactical relationship are grouped together, the merging process first examines productions for the strings in the same cluster. Productions corresponding to a string are compared with those for its successors in the labelled MST. Productions for different clusters are then examined according to the inference tree for further consolidation. The heuristic procedure for merging productions from different cluster is similar to that for merging productions in a cluster. Some of the heuristic tactics applied in Sec. 5.3.2.2. can also be applied here.

Example 5: Let sample S_1 and its initial CNFPG G_1 be defined as in Ex. 4. The merging process starts with the productions that generate the root and its immediate successors. According to Rule 5 in Sec. 5.4.2.1., $A \rightarrow B_1B_2$ and $A \rightarrow B_3B_2$ are merged into $A \rightarrow B_1B_2$ (3/8), and Rule 4, $B_1 \rightarrow c|B_1B_4$ (2/3, 1/3) is created. Similarly, B_5, and then B_6 are replaced by B_1. After all consolidations, we obtain:

$A \rightarrow B_1B_2$ (1/2), $B_1 \rightarrow c|B_1B_4$ (1/2, 1/2), $B_2 \rightarrow b$, $B_4 \rightarrow$ a.

Assuming this is the best set of productions for S_1, then the merging process for the cluster stops.

Example 6: Let $S = \{S_1, S_2\}$ be a p-sample, S_1 is defined as in Ex. 5., and $S_2 = \{bbb, bbab, bbaab, bbaaab\}$ with associated probabilities $\{1/4, 1/8, 1/16, 1/16\}$. By constructing and merging processes similar to that in Ex. 4 and Ex. 5, assuming the final CNFPG for S_2 has the productions:

$A \rightarrow D_1D_1$ (1/2), $D_1 \rightarrow b|D_1D_1|D_1D_2$ (1/4, 1/4, 1/2), $D_2 \rightarrow a$.

Now we merge productions from the two p-grammars in an heuristic manner. According to Rule 1 in Sec. 5.4.2.1, D_2 is replaced by B_4, and Rule 4, $A \rightarrow D_1D_1|B_1B_2$ (1/2, 1/2) is created. Assuming no more productions can be merged, then we obtain the solution grammar with the productions:

$A \rightarrow B_1B_2|D_1D_2$ (1/2, 1/2), $B_1 \rightarrow c|B_1B_4$ (1/2, 1/2), $B_2 \rightarrow b$, $B_4 \rightarrow a$, $D_1 \rightarrow b|D_1D_1|D_1B_4$ (1/4, 1/4, 1/2).

5.5. COMPARISON

In the previous chapters, constructive methods had been proposed for the inference of finite-state and Chomsky normal form p-grammars from a p-sample. In this section, the constructive methods are compared with other constructive methods. Based on the common features of constructive methods described in Sec. 5.1., their generality, complexity, limitations and completeness will be compared. The constructive method proposed for the inference of finite-state grammars is compared with the three most significant existing constructive methods for finite-state grammars /3/,/20/, /52/.

In the analysis of sample structure, the proposed method uses a cluster analysis technique based on a labelled MST while the other three methods /3/, /13/, /52/ use formal derivative and k-tails (Sec. 3.2.). The clustering method uses dissimilarities as well as common substrings to analyze sample structure, while the formal derivative and k-tails methods only sequentially search for common characters of sample strings to decide the informational relationship among sample strings. The clustering method is more general and less limited than the formal derivative. The clustering method will produce a reasonable result for any set of sample strings, while the formal derivative and k-tails methods are inherently limited.

In the construction of finite-state grammars, the proposed method uses the inference tree as a guide while the other three methods /3/, /13/, /52/ use an arbitrary subset of sample strings, or every string in the sample, guided by the results of formal derivative to construct grammars. The former is more efficient than the later, since it uses organized subsets of sample strings and the inference tree to construct grammars with less effort.

In merging productions, the proposed method uses the relationship between two productions and the concept of grammatical complexity to merge related productions and to simplify the grammar, while the three methods /3/,/13/,/52/ use a derived grammar or merge equivalent nonterminals to produce grammars. The former is more complex, complete and general, but less limited than the later.

The proposed method searches for a solution grammar which has least complexity and generates an acceptable language, while the three methods /3/,/13/,/52/ obtain a solution grammar without the complexity criterion.

The constructive method proposed for the inference of Chomsky normal form p-grammars is compared with the three most significant constructive methods for the subsets of context-free languages /14/-/16/,/24/,/61/,/62/, and the two most significant constructive methods for context-free p-languages /11/,/46/. The results of the comparison are presented separately below.

Comparing the proposed method to the three constructive methods for the subsets of context-free languages /14/-/16/,/24/,/61/,/62/ produces three main conclusions.

In the analysis of sample structure, the cluster analysis method uses dissimilarities as well as substring relationships to classify sample strings, while the three methods use substring and structural relationships to analyze sample structure. The former is more general, complex and complete, but less limited than the later, since the clustering method considers more factors than the three methods /3/, /13/,/52/ in analyzing sample structure and the clustering method is not limited to analyzing subsets of context-free languages.

The proposed method is more general than the three methods /3/,/13/,/52/, since it is not limited to the inference of subsets of context-free languages.

The results of comparing the proposed method to the two constructive methods for context-free p-languages /11/, /46/ are summarized below:

The clustering method considers various differences between strings to classify the sample strings, while the two methods only use a simple substring relationship to analyze sample structure. In the construction of context-free p-grammars, the proposed method uses the inference tree as a guide to construct grammars and to merge productions, while the two methods start with a simple grammar and use pattern matching techniques or heuristic search of common substrings to merge productions. The proposed method is more general, complex, particular and complete than the two methods, since it uses a well formed framework to construct grammars.

The proposed method has a more reasonable difference measure of p-languages than those two methods have (Cf. Sec. 4.4.).

5.6. COMPUTATIONAL CONSIDERATIONS

In this section, we discuss the computational difficulties inherent in the application of the constructive method developed in this chapter.

In the literature, all examples for demonstrating the performance of constructive procedures either involve a small number of sample strings or are designed for a very specific type of grammar. Since the field of grammatical inference is still in its infancy, few restrictions are placed on solution grammars and no uniform standard of performance exists. Little optimization is possible. In addition, existing constructive methods cannot generate an adequate grammar even when the sample contains a moderate number of strings, much less when a few hundred strings are in the sample. There is no common ground for comparing two different constructive methods. In most cases, constructive methods are developed for some specific purposes and under different assumptions and restrictions. A few examples are not enough to show one constructive procedure is uniformly better than another constructive procedure.

As mentioned before, the constructive method developed in this thesis consists of three main components: Analyzing the syntactic structure of sample strings, constructing candidate grammars and testing acceptance criteria. In analyzing the syntactic structure of sample strings, the clustering method in this thesis was developed particularly for handling a general situation and a large number of sample strings.

There is a problem in obtaining a real and reasonably large set of sample strings. In selecting a real sample, the sample space is the power set of a terminal set, and strings of small length are more likely to be selected than long strings. Thus, the syntactic structure of a sample in practice is not obvious. On the other hand, the artificial samples used to demonstrate constructive methods in the literature are usually taken from an hypothesized source language whose syntactic structure is apparent.

Since the construction of candidate grammars is guided by an inference tree, the number of grammars being constructed equals the number of clusters formed for the sample. For a large number of sample strings, a relatively large number of clusters will be formed. The problem of merging related productions to generate candidate grammars becomes more difficult when the number of initial grammars as well as productions increase. The nature of the difficulty lies in both the number of comparisons needed to find a merger, and the number of productions involved in the consolidation after a merger is adopted. In addition, the problem of assigning production probabilities becomes serious when the consolidations following mergers involve many productions.

The acceptance criterion depends on the type of mergers used and requires knowledge of the language generated by a grammar. Finding the probabilistic language generated by an inferred p-grammar requires a parser. A probabilistic finite-state automaton is needed for parsing a finite-state p-language, and a probabilistic push down automaton is needed for parsing a context-free p-language. Constructing and updating these parsers are complicated tasks. Besides, when the resulting p-grammar is not acceptable, it is difficult to decide what should be changed first, dissimilarity measure, clustering techniques, or merging strategies. Defining heuristic strategies to optimizing an acceptance criterion is beyond the scope of this thesis.

The entire constructive method is not easy to program. The main difficulties are in the selection of proper data structures for a p-grammar, in the construction of efficient parsers for recognizing strings, and in the implementation of heuristic strategies for optimizing an acceptance criterion.

5.7. SUMMARY

In this chapter we have discussed the problem of constructing p-grammars from a p-sample. A method of assigning production probabilities is described. The procedures for inferring finite-state and Chomsky normal form p-grammars from a p-sample are presented, and compared with other constructive methods in the same category. In general, the proposed method is more efficient, complex, general and complete than other existing constructive methods.

The constructive methods proposed are based on the techniques discussed in the previous chapters. The method consists of two steps and based on an inference tree, construct an initial p-grammar exactly describing the sample strings, then merge related productions to generate candidate p-grammars. Situations in which two productions may be merged together have been discussed, and the necessary consolidations for each situation have been described. Two examples have been presented to explain how the constructive methods work for the inference of finite-state and Chomsky normal form p-grammars. The computational difficulties for the application of the constructive method have been discussed.

CHAPTER VI

SUMMARY, CONCLUSIONS AND FUTURE RESEARCH

This chapter summarizes the main results of the thesis and discusses possibilities for future research.

6.1. SUMMARY AND CONCLUSIONS

This thesis is concerned with studying constructive procedures for inferring a simple and acceptable p-grammar from a p-sample. A heuristic approach is taken to grammatical inference. Clustering methods, grammatical complexity measures, difference measures and statistical tests are used as the tools for developing constructive methods for grammatical inference. The objective was to develop general and efficient constructive methods for inferring finite-state and Chomsky normal form p-grammars that generate not only the p-sample but also a language that resembles the p-sample in some sense.

The most significant models, concepts and techniques available in the literature are reviewed and briefly discussed in Chapter II. Chapter III introduces a procedure for analyzing sample structure. This procedure is one of the major contributions of the thesis. Five measures of dissimilarity between strings based on the minimal cost of a sequence of edit operations are defined and compared and their properties are investigated. Examples are also presented to show the effects of different dissimilarity measures in discriminating among strings having different inherent relationships. A labelled MST for a set of sample strings which leads to an inference tree are the fundamental notions used to establish structure. A clustering algorithm based on the MST which uses the notions of "common substrings" and "length" to define inconsistent edges is proposed. Seven interpretations of inconsistent edges are defined. The removal of inconsistent edges generates the inference tree. The main contributions of Chapter III are the developments of dissimilarity measures and clustering algorithms for analyzing sample structure, none of which have been proposed in the literature. The resulting inference tree is unique to this thesis and is one of the key factors used for developing general and computationally efficient constructive methods in Chapter V. The procedure is demonstrated to be more general than methods for analyzing sample structure in the literature.

Grammatical complexity measures and acceptance criterion are investigated in Chapter IV. The problem of defining solution grammars is carefully examined, and a definition of solution grammar is adopted for this thesis. A size measure is suggested for evaluating the performance of dissimilarity measures. A complexity measure based on information theory is defined for optimizing the selection of a grammar. Several difference measures for languages are defined and investigated, especially those based on information theory. For the first time, the Kolmogrov-Smirnov maximum deviation statistic is applied to test the appropriateness of a p-language for a p-sample, and compared with the chi-square goodness of fit test. The main contributions of Chapter IV are the development of difference measures for p-languages and the suggestions of statistical tests and complexity measures for grammatical inference. The difference measures proposed are more realistic than those in the literature. The development of constructive methods depends heavily on the techniques in Chapter IV.

Procedures for actually constructing finite-state and context-free p-grammars are proposed in Chapter V. The constructive procedure for finite-state p-grammars is similar to that for Chomsky normal form p-grammars, and consists of two parts: construction of an initial p-grammar, and merging related productions to generate candidate p-grammars. The construction is guided by the inference tree of Chapter III and the merging rules optimize the complexity measure of Chapter IV. Rules for assigning production probabilities are described, and the techniques presented in previous chapters are integrated into constructive methods which are compared with constructive methods in the literature. The main contribution of Chapter V is the synthesis of the constructive method itself. The merging rules are unique to this thesis. Examples are presented to demonstrate the constructive process. The constructive method proposed here is unique in the procedure for analyzing sample structures, the use of difference measures for p-languages, the type of statistic acceptance test and the heuristic strategies for constructing initial and candidate p-grammars. The proposed constructive methods are more general, complex, complete and less limited than other constructive methods.

The nature of this research precludes mathematical proofs of several claims made in this thesis. Introducing enough grammatical and mathematical structure to prove theorems would have destroyed the very problem under investigation. The true value of this thesis will only become apparent when applied to a real data base. Unfortunately, the generation of such a data base requires image processing equipment not currently available at M.S.U.

6.2. FUTURE RESEARCH

The following topics are suggested for future research in the area of grammatical inference.

(1) In measuring dissimilarity between strings, it is evident that the ability to discriminate among sample strings varies considerably with the dissimilarity measure chosen. The investigation of the effect of different dissimilarity measures on discrimination, and the optimization of dissimilarity measures are important topics for future research.

(2) The nonoverlapping hierarchical clustering algorithm developed in this thesis is based on a labelled MST.

The study of other clustering methods for analyzing sample structure is needed. In particular, the development of overlapping clustering methods for the inference of ambiguous grammars and the effect of different clustering methods in grammatical inference are important topics.

(3) The chi-square goodness of fit and Kolmogrov-Smirnov maximum deviation tests have some disadvantages in grammatical inference. Other statistics need to be investigated, particularly non-parametric statistics.

(4) In forming candidate p-grammars from initial p-grammars, the effect of different heuristic strategies such as the order of merging productions should be investigated. The same can be said for merging grammars for different clusters.

APPENDIX

APPENDIX A

CLUSTERING ALGORITHM

Given a subtree of the labelled MST consists of node N with immediate successors $N_1, N_2, \ldots N_n$, and immediate predecessor N' of N. Let

k_x = length of maximal substrings common to N and N_x.

u_x = Number of symbols not used in both N and N_x.

d_x = Weight of edge (N, N_x).

$\ell(N)$ = length of node N.

The procedures P_1 and P_2 for obtaining an inference tree from a labelled MST are given below.

Procedure P_1:

* Stack 1 : Labelled MST

* Stack 2 : Inference Tree

* Stack 3 : Roots of subtrees not being classified

* Stack 4 : Immediate successors of N in lexicographic order

```
1.  Get Root N from Stack 1
2.  Set $N' \leftarrow 0$
3.  Push N on Stack 2
4.  Set $j \leftarrow 0$
5.  Get immediate successors $N_1, N_2, \ldots N_n$ of N from Stack 1
```

```
6.  Push $N_1, N_2, \ldots N_n$ on Stack 4
7.  Call $P_2(N,N',j)$
8.  If $j \neq 0$, set $N' \leftarrow N$, $N \leftarrow N_j$, go to 3
9.  If Stack 3 is empty, stop
10. Pop stack 3, get N, go to 2
```

Procedure P_2 (N,N',j)

```
*  Stack 3 : Roots of subtrees not being classified
*  Stack 4 : Immediate successors of N in lexicographic order

1.  If Stack 4 is empty, return, else pop Stack 4, get $N_x$
2.  If $\ell(N)=\ell(N_x)$, go to 10
3.  If $N'=0$ go to 5
4.  If $\ell(N)>\max(\ell(N'),\ell(N_x))$ or $\ell(N)<\min(\ell(N'),\ell(N_x))$, go to 10
5.  If $j = 0$ go to 12
6.  If $k_x = 0$ or $k>k_x$, go to 10
7.  If $k_x>k$ or $u>u_x$, go to 11
8.  If $u_x>u$ or $d_x>d$, go to 10
9.  If $d>d_x$, go to 11
10. Push $N_x$ on Stack 3, go to 1
11. Push $N_j$ on Stack 3
12. Set $k \leftarrow k_x$, $u \leftarrow u_x$, $d \leftarrow d_x$, $j \leftarrow x$, go to 1
```

APPENDIX B

B1. RULES FOR MERGING FINITE-STATE PRODUCTIONS

Situation	Pairs of Candidate Productions
(1)	$(A \rightarrow aB,\ C \rightarrow aB)$, $(A \rightarrow aB,\ B \rightarrow aB)$, $(A \rightarrow a,\ B \rightarrow a)$
(2)	$(A \rightarrow aB,\ B \rightarrow aA)$
-(3)	$(A \rightarrow aB,\ B \rightarrow aC)$, $(A \rightarrow aB,\ B \rightarrow a)$
(4)	$(A \rightarrow aB,\ A \rightarrow bC)$, $(A \rightarrow aB,\ A \rightarrow bB)$, $(A \rightarrow aA,\ A \rightarrow bB)$, $(A \rightarrow aB,\ A \rightarrow b)$, $(A \rightarrow a,\ A \rightarrow b)$
(5)	$(A \rightarrow aB,\ A \rightarrow aC)$, $(A \rightarrow aA,\ A \rightarrow aC)$
(6)	$(A \rightarrow aB,\ B \rightarrow bC)$

B2. RULES FOR MERGING CHOMSKY NORMAL FORM PRODUCTIONS

Situation	Pairs of Candidate Productions
(1)	$(A \rightarrow BC,\ D \rightarrow BC)$, $(A \rightarrow BA,\ D \rightarrow BA)$, $(A \rightarrow a,\ B \rightarrow a)$, $(B \rightarrow AA,\ C \rightarrow AA)$, $(A \rightarrow AB,\ B \rightarrow AB)$, $(A \rightarrow AA,\ B \rightarrow AA)$
(2)	$(A \rightarrow BC,\ C \rightarrow BA)$, $(A \rightarrow CB,\ C \rightarrow AB)$
(3)	$(D \rightarrow BA,\ A \rightarrow BC)$, $(D \rightarrow AB,\ A \rightarrow CB)$, $(A \rightarrow BC,\ D \rightarrow AA)$, $(A \rightarrow BC,\ B \rightarrow CC)$, $(A \rightarrow BC,\ C \rightarrow BB)$, $(A \rightarrow a,\ B \rightarrow AA)$
(4)	$(A \rightarrow BC,\ A \rightarrow BD)$, $(A \rightarrow CB,\ A \rightarrow DB)$, $(A \rightarrow AB,\ A \rightarrow AC)$, $(A \rightarrow BA,\ A \rightarrow CA)$
(5)	$(A \rightarrow BC,\ A \rightarrow DE)$, $(A \rightarrow BC,\ A \rightarrow CB)$, ...etc.
(6)	$(A \rightarrow BC,\ B \rightarrow CD)$, $(A \rightarrow BC,\ C \rightarrow DE)$, ...etc.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. M.R. Anderberg, "Cluster Analysis for Applications," Academic Press, Inc., New York, 1973.

2. A.W. Biermann and J.A. Feldman, "A survey of results in grammatical inference," in Frontiers of Pattern Recognition, pp. 31-54, ed. by S. Watanaba Academic Press, New York, 1972.

3. A.W. Biermann and J.A. Feldman, "On the synthesis of finite-state machines from samples of their behavior," IEEE Trans. Comput., vol. C-21, pp. 592-597, June 1972.

4. Z.W. Birnbaum, "Numerical Tabulation of Distribution of the Distribution of Kolmogrov's Statistic for Finite Sample Size," Journal of the American Statistical Association, 47, 425-441, 1952.

5. M. Blum, "On the size measure," Inf. and Cont., vol. 11, pp. 257-265, 1967.

6. M. Blum, "A Machine-Independent theory of the complexity of recursive functions," J. ACM, vol. 14, pp. 322-336, 1967.

7. T.L. Booth, "Probabilistic representation of formal languages," in Proc. 10th IEEE Ann. Switching and Automata Theory, 1969.

8. T.L. Booth and Y.T. Chien, Computing: Fundamentals and Applications. Santa Barbara, Calif., Hamilton, 1974.

9. T.L. Booth and R.A. Thompson, "Applying probability measures to abstract languages," IEEE Trans. Comput., vol. C-22, pp. 442-450, May 1973.

10. J.A. Brzozowski, "Derivatives of Regular Expressions," J.ACM, vol. 11, No. 4, pp. 481-494, Oct. 1964.

11. C.M. Cook, "A Cost Function for Concept Formation," "Experiments in Grammatical Inference," "Grammatical Inference by Heuristic Search", Comput. Sci. Center, Univ. Maryland, College Park, Tech. Rep., TR-212, 1972, TR-257, 1973, TR-287, 1974.

12. H. Cramer, Mathematical Models of Statistics. Princeton, N.J.: Princeton Univ., 1946.

13. A. Cremers and O. Mayer, "On matrix languages," Inf. and Cont., vol. 23, pp. 86-96, 1973.

14. S. Crespi-Reghizzi, "An Effective Model for Grammar Inference," in Proc. IFIP Congr., 1971.

15. S. Crespi-Reghizzi, "Reduction of Enumeration in Grammar Acquisition," presented at the 2nd Int. Joint Conf. Artificial Intelligence, London, England, Sept. 1-3, 1971.

16. S. Crespi-Reghizzi, M.A. Melkanoff, and L. Lichten, "The use of Grammatical Inference for Designing Programming Languages," Commun.ACM, vol. 16, pp. 83-90, Feb. 1973.

17. R.C. Dubes, "Information Compression, Structure Analysis and Decision Making with a Correlation Matrix," AD 826811 Michigan State University, E.Lansing, Michigan, 1970.

18. A.W.F. Edwards and L.L. Cavalli-Sforza, "Reconstruction of evolutionary trees," Phe. and Phy. Classifications, pp. 67-76, 1964.

19. C.A. Ellis, "Probabilistic languages and automata," Ph.D. dissertation, Univ. of Illinois, Urbana, Illinois, 1969.

20. J.A. Feldman, "First Thought on Grammatical Inference," Standford, Art. Int. Project Memo. No. 55, Standford University, Standford, California 1967.

21. J.A. Feldman, "Some Decidability Results on Grammatical Inference and Complexity," Inf. and Cont., vol. 20, pp. 244-262, 1972.

22. J.A. Feldman and D. Gries, "Translator writing systems," Comput. ACM, vol. 11, pp. 77-113, Feb. 1968.

23. J.A. Feldman and P. Shields, "On Total Complexity and the existence of best programs," Comput. Sci. Dep., Stanford Univ., Stanford, Calif., Tech. Rep. CS-255, 1972.

24. J.A. Feldman, J. Gips, J.J. Horning and S. Reder, "Grammatical Complexity and Inference," Comput. Sci. Dep., Standford Univ., Standford, Calif., Tech. Rep. CS-125, 1969.

25. K.S. Fu, Syntactic Methods in Pattern Recognition, New York Academic, 1974.

26. K.S. Fu, "Stochastic automata, stochastic languages and pattern recognition," J. Cybernetics, vol. 1, no. 3, pp. 31-49, 1971.

27. K.S. Fu and T.L. Booth, "Grammatical Inference: Introduction and Survey - Part I," IEEE Trans. Syst., Man, Cybern., vol. SMC-5, pp. 95-111, Jan. 1975.

28. K.S. Fu and T.L. Booth, "Grammatical Inference: Introduction and Survey - Part II," IEEE Trans. Syst., Man, Cybern., vol. SMC-5, pp. 409-423, July 1975.

29. K.S. Fu and T. Huang, "Stochastic grammars and languages," Int. J. Comput. Inform. Sci., vol. 1, no. 2, pp. 135-170, 1972.

30. E.M. Gold, "Language Identification in the Limit," Inf. and Cont., vol. 10, pp. 447-474, 1967.

31. J.C. Gower, "A comparison of some methods of cluster analysis," Biometrics, vol. 23, pp. 623-637, 1967.

32. J. Gruska, "Some classifications of context-free languages," Inf. and Cont., vol. 14, pp. 152-179, 1969.

33. J. Hartmanis and R.E. Stearns, Algebric Structure Theory of Sequential Machines, Prentice-Hall, 1966.

34. J.E. Hopcroft and J.D. Ullman, "Formal Languages and Their Relation to Automata." Reading, Mass., Addison-Wesley, 1969.

35. J.J. Horning, "A Procedure for Grammatical Inference," Proc. IFIP Congress 71, Ljubljana.

36. J.J. Horning, "A Study of Grammatical Inference," Ph.D. Thesis, Standford University, Standford, Calif.

37. N. Jardine and R. Sibson, "Mathematical Taxonomy," John Wiley, New York, 1970.

38. S.C. Johnson, "Hierarchical Clustering Schemes," Psychometrika, vol. 32, pp. 241-254, 1967.

39. L. Kanal, "Patterns in Pattern Recognition," IEEE Trans. Inf. Theory, vol. IT-20, pp. 697-722, 1974.

40. A. Kolmogrov, "Confidence Limits for an Unknown Distribution Function," Annals of Mathematical Statistics, 12, 461-463, 1941.

41. R.R. Korfhage, Discete Computational Structures, pp. 101-129, Acad. Press, N.Y., 1974.

42. J.B. Kruskal, "On the shortest spanning subtree of a graph and a travelling salesman problem," Proc. Amer. Math. Soc., 7, 48-50, 1956.

43. W. Kuich, "On the entropy of context-free languages," Inf. and Cont., vol. 16, pp. 173-200, 1970.

44. B.W. Lindgren, Statistical Theory, Collier-Macmillan Ltd., London, 1968.

45. H. Loberman and A. Weinberger, "Formal procedures for connecting terminals with a minimum total wire length," J. ACM, vol. 4, pp. 428-237, 1957.

46. F.J. Maryanski, "Inference of Probabilistic Grammars," Ph.D. dissertation, Dep. Elec. Eng. Comput. Sci., Univ. Conn., Storrs, July 1974.

47. F.J. Massey, "A Note on the Power of a Non-Parametric Test," Annals of Mathematical Statistics, 21, 440-443, 1950 (See also 23 (1952), 637-638).

48. N.J. Nillson, Problem-solving Methods in Artificial Intelligence, McGraw-Hill Book Company, New York, 1971.

49. T. Okuda, E. Tanaka and T. Kasai, "A Method for the Correction of Garbled Words Based on the Levenshtein Metric," IEEE Trans. on Comput., vol. C-25, No. 2, pp. 172-178, 1976.

50. T.W. Pao, "A solution of the syntactical induction-inference problem for a non-trivial subset of context-free languages," Moore Sch. Elec. Eng., Univ. Pennsylvania, Philadelphia, Interim Tech. Rep. 69-19, 1969.

51. E. Parzen, Modern Probability Theory and its Applications, John Wiley, New York, 1960.

52. A. R. Patel, "Grammatical inference for probabilistic finite-state languages," Ph.D. dissertation, Dept. Elec. Eng. Comput. Sci., Univ. of Conn., Storrs, 1972.

53. R.C. Prim, "Shortest Connection Networks and Some Generalizations," Bell Systems Technical Journal, Vol. 36, pp. 1389-1401, 1957.

54. P.S. Rosenbaum, "A Grammar base question-answering procedure," Comput. ACM, vol. 10, pp. 630-635, 1967.

55. G.J.S. Ross, "Single Linkage Cluster," Algorithms as 13-15, Applied Statistics, vol. 18, pp. 50-54, 1969.

56. A. Salomma, "On the index of a context-free grammar and languages," Inf. and Cont., vol. 14, pp. 474-477, 1969.

57. A. Salomma, "Probabilistic and weighted grammars," Inf. and Cont., vol. 15, pp. 529-544, 1969.

58. E.S. Santos, "Probabilistic grammars and automata," Inf. and Cont., vol. 21, pp. 27-47, 1972.

59. C.E. Shannon and W. Weaver, The Mathematical Theory of Communication, The University of Illinois Press, Urbana, 1963.

60. M.J. Shepherd and A.J. Willmott, "Cluster Analysis on the Atlas Computer," Comp. J., II, pp. 57-62, 1968.

61. R.J. Solomonoff, "A Formal Theory of Inductive Inference - Part I, Part II," Inf. and Cont., vol. 7, pp. 1-22, 224-254, 1964.

62. R.J. Solomonoff, "A New Method for Discovering the Grammars of Phrase Structure Languages," in Information Processing. New York: UNESCO, 1959.

63. S. Soule, "Entropies of probabilistic grammars," Inf. and Cont., vol. 25, pp. 57-74, 1974.

64. R.A. Thompson, "Determination of probabilistic grammars for functionally specified probability-measure languages," IEEE Trans. Comput., vol. C-23, pp. 603-614, June 1974.

65. M.G. Thomason, "Stochastic syntax-directed translation schemator for correction of errors in context-free languages," IEEE Trans. Comput., vol. C-24, pp. 1211-1216, Dec. 1975.

66. R.A. Wagner and M.J. Fischer, "The String-to-String Correction Problem," J. ACM, vol. 21, pp. 168-173, 1974.

67. R.M. Wharton, "Grammatical Inference and Approximation," Ph.D. Thesis, University of Toronto, Toronto, Canada, 1973.

68. R.M. Wharton, "Approximate Language Identification," Inf. and Cont., vol. 2, pp. 236-255, 1974.

69. C.T. Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters," IEEE Trans. Comput., vol. C-20, No. 1, pp. 68-86, Jan. 1971.