ERROR-CORRECTING PARsing
FOR SYNTACTIC PATTERN RECOGNITION

C. T. Lu
and
K. S. Fu

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

TR-EE 77-31
August 1977

This work was supported by the AFOSR Grant 74-2661.
The problem of modeling, analysis and reconstruction of noisy and/or distorted syntactic patterns is studied. Segmentation errors and primitive extraction errors can be treated as syntax errors and defined in terms of language transformation rules. Three types of error transformations are defined on strings, namely substitution, insertion and deletion. Consequently, the parser constructed according to the grammar generating the strings and the three types of transformations is called the error-correcting parser.
correcting parser. This technique is also extended to tree languages. In formulating error-correcting tree automata (ECTA), five types of error-transformations on trees are defined, namely, substitution, split, stretch, branch and deletion. By way of using language transformations, the distance between two sentences can be determined. A definition of distance between a sentence and a language is proposed. Based on this definition, a clustering procedure is proposed, where error-correcting parsers...
ERROR-CORRECTING PARSING
FOR SYNTACTIC PATTERN RECOGNITION

S. Y. Lu and K. S. Fu

TR-EE 77-31
August, 1977

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DDC
This technical report has been reviewed and is approved for public release IAF AFR 190-12 (7b).
Distribution is unlimited.
A. D. BLOSE
Technical Information Officer

+This work was supported by the AFOSR Grant 74-2661.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>x</td>
</tr>
<tr>
<td>CHAPTER 1 - INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Purpose</td>
<td>1</td>
</tr>
<tr>
<td>1.2 The Recognition of Noisy and Distorted Patterns</td>
<td>1</td>
</tr>
<tr>
<td>1.3 Survey of Error-Correcting Parsing</td>
<td>7</td>
</tr>
<tr>
<td>1.4 Thesis Organization</td>
<td>13</td>
</tr>
<tr>
<td>CHAPTER 2 - ERROR-CORRECTING PARSING FOR STRING LANGUAGES</td>
<td>16</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>16</td>
</tr>
<tr>
<td>2.2 Minimum Distance Error-Correcting Parsing for Context-Free Languages</td>
<td>18</td>
</tr>
<tr>
<td>2.2.1 A Similarity Measure for Syntactic Patterns</td>
<td>19</td>
</tr>
<tr>
<td>2.2.2 A Minimum-Distance Error-Correcting Parser</td>
<td>23</td>
</tr>
<tr>
<td>2.2.3 A Minimum-Distance Classifier for Noisy Patterns</td>
<td>26</td>
</tr>
<tr>
<td>2.3 Error-Correcting Parsing for Context-Free Programmed Languages</td>
<td>30</td>
</tr>
<tr>
<td>2.3.1 Context-Free Programmed Grammar</td>
<td>30</td>
</tr>
<tr>
<td>2.3.2 Error-Correcting Parsing Algorithm</td>
<td>33</td>
</tr>
<tr>
<td>2.4 Error-Correcting Parsing on a Stochastic Model</td>
<td>37</td>
</tr>
<tr>
<td>2.4.1 A Stochastic Model for Syntax Error</td>
<td>39</td>
</tr>
<tr>
<td>2.4.2 Maximum-Likelihood Error-Correcting Parser (MLECP)</td>
<td>43</td>
</tr>
<tr>
<td>2.4.3 Bayes Classification of Noisy Patterns</td>
<td>46</td>
</tr>
<tr>
<td>2.4.4 An Illustrative Example</td>
<td>48</td>
</tr>
<tr>
<td>2.4.5 Sequential Classification of Noisy Patterns</td>
<td>57</td>
</tr>
<tr>
<td>CHAPTER 3 - ERROR-CORRECTING TREE AUTOMATA</td>
<td>63</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>63</td>
</tr>
<tr>
<td>3.2 Definitions</td>
<td>64</td>
</tr>
<tr>
<td>3.3 Structure-Preserved Error-Correcting Tree Automaton (SPECTA)</td>
<td>67</td>
</tr>
<tr>
<td>3.3.1 Minimum-Distance SPECTA</td>
<td>68</td>
</tr>
<tr>
<td>3.3.2 Maximum-Likelihood SPECTA</td>
<td>71</td>
</tr>
<tr>
<td>3.3.3 Application of SPECTA to LANDSAT Data Interpretation</td>
<td>78</td>
</tr>
<tr>
<td>3.4 Generalized Error-Correcting Tree Automaton</td>
<td>92</td>
</tr>
<tr>
<td>3.4.1 Distance on Trees</td>
<td>92</td>
</tr>
<tr>
<td>3.4.2 The Formulation of GECTA</td>
<td>96</td>
</tr>
<tr>
<td>3.4.3 An Illustrative Example on Character Recognition</td>
<td>113</td>
</tr>
</tbody>
</table>
CHAPTER 4 - CLUSTERING ANALYSIS FOR SYNTACTIC PATTERNS

4.1 Introduction ................................................. 120
4.2 Sentence-to-Sentence Clustering Algorithms .................... 121
  4.2.1 A Nearest Neighbor Classification Rule .................. 121
  4.2.2 The Cluster Center Techniques .......................... 123
4.3 An Illustrative Example ..................................... 124
  4.3.1 Data Preparation ....................................... 124
  4.3.2 Experiments ........................................... 127
4.4 A Proposed Nearest Neighbor Syntactic Recognition Rule ....... 145
4.5 A Clustering Procedure for Syntactic Patterns ................ 146
  4.5.1 The Algorithm ......................................... 146
  4.5.2 An Experiment ......................................... 148
4.6 Conclusions and Remarks .................................... 160

CHAPTER 5 - A SYNTACTIC APPROACH TO TEXTURE ANALYSIS ........ 162

5.1 Introduction ................................................ 162
5.2 A Syntactic Model for Texture Analysis ......................... 164
  5.2.1 The Primitive ......................................... 164
  5.2.2 The Window ............................................ 164
  5.2.3 The Tree Representation ................................ 171
  5.2.4 The Tree Grammar ..................................... 172
5.3 Illustrative Examples of Texture Synthesis ..................... 177
  5.3.1 Regular Tesselation .................................... 183
  5.3.2 Irregular Tesselation .................................. 191
  5.3.3 Random Pattern ........................................ 194
5.4 Texture Discrimination .................................... 201
  5.4.1 Data Preparation ...................................... 201
  5.4.2 Discrimination Grammars ............................... 201
  5.4.3 Error-Correcting Parsing ............................... 205
  5.4.4 Computation Result .................................... 205
5.5 Remarks .................................................... 206

CHAPTER 6 - SUMMARY OF RESULTS AND SUGGESTIONS FOR FURTHER RESEARCH .... 208

6.1 Summary of Results and Conclusions ........................ 208
6.2 Suggestions for Further Research ............................. 210

BIBLIOGRAPHY ................................................. 212

APPENDICES

APPENDIX A ................................................. 220
APPENDIX B ................................................. 222
APPENDIX C ................................................. 225
APPENDIX D ................................................. 233
APPENDIX E ........................................... 250
APPENDIX F ........................................... 272
APPENDIX G ........................................... 308
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>$G_S$, $G_M$, $G_A$, $G_T$ for submedian, median, acrocentric, and telocentric, respectively.</td>
<td>52</td>
</tr>
<tr>
<td>2.2</td>
<td>Probabilities of Terminal Error Transformations.</td>
<td>52</td>
</tr>
<tr>
<td>2.3</td>
<td>Maximum Number of Errors Allowed in Substrings of cbabdbabcbabdb when parsed by $G_M^T$ with Lower Bound $0.001(.5)^{-1}$.</td>
<td>55</td>
</tr>
<tr>
<td>2.4</td>
<td>Classification Results from 200 Strings (112 of them are Erroneous)</td>
<td>60</td>
</tr>
<tr>
<td>2.5</td>
<td>Classification Results from 200 Correct Strings.</td>
<td>60</td>
</tr>
<tr>
<td>2.6</td>
<td>Sequential Error-Correcting Classification</td>
<td>62</td>
</tr>
<tr>
<td>4.1</td>
<td>The 51 Character Patterns and Their Representations</td>
<td>134</td>
</tr>
<tr>
<td>4.2</td>
<td>The Three Iteration Results Using Algorithm 4.2</td>
<td>142</td>
</tr>
<tr>
<td>4.3</td>
<td>An Inferred Grammar for Pattern 1.</td>
<td>150</td>
</tr>
<tr>
<td>4.4</td>
<td>The Result Clusters of the 51 Characters</td>
<td>153</td>
</tr>
<tr>
<td>4.5</td>
<td>New Rules Added to the Original Grammars</td>
<td>155</td>
</tr>
<tr>
<td>4.6</td>
<td>The Grammar $G_{51}$</td>
<td>157</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure

1.1  Block Diagram of a Syntactic Pattern Recognition System  ...  3
1.2  Syntax Analysis Using a Parser (a) and an Error-Correcting Parser (b)  ...  8

2.1  A Graphic Interpretation of Metric W.  ...  22
2.2  A Minimum-Distance Classifier for Noisy Patterns  ...  27
2.3  Analysis of abc with Respect to $G_p$  ...  32
2.4  Syntax Analysis of aabc with Respect to $G_p$  ...  34
2.5  The Analysis of aabc when n=1  ...  38
2.6  The Stochastic Deformation Model Described by a Lattice  ...  42
2.7  Bayes Classification System for Noisy Strings  ...  49
2.8  Typical Chromosome Patterns and Their String Representation  ...  50
2.9  Comparison of Number of Items in the Parse List of String cbabdbdbabdbabdbab Parsed by $G^*_p$ with and without Preset Lower Bound  ...  54
2.10  cpu Time vs. String Length  ...  56
2.11  Accuracy vs. Parsing Time of Sequential Classifier and Non-Sequential Classification  ...  59

3.1  Vertical Line Patterns  ...  73
3.2  A Tree Structure  ...  73
3.3  Tree Representation of Pattern (b)  ...  73
3.4  A Noisy Pattern (a), and its Tree Representation, $\alpha'$ (b)  ...  74
3.5  Transition Table of Tree $\alpha'$  ...  75
3.6  The Generation of $\alpha$, the Correction of $\alpha'$  ...  76
Figure
3.7  Pointwise Classified Highway Data Obtained from Grand Rapids, Michigan. .......................... 79
3.8  Divided Highway Map of Northern Part of Grand Rapids. ..................................................... 80
3.9  Nodes in Tree Domain $D_H$ and Their Corresponding Positions in the $8 \times 8$ Array ............. 83
3.10 Window-by-Window Correcting Process ..................................................................................... 86
3.11 Flow Chart of Highway Recognition Using SPECTA. ............................................................... 87
3.12 SPECTA Processing Result of the Grand Rapids Area ............................................................. 88
3.13 Pointwisely Classified Data of the Lafayette Area .................................................................... 89
3.14 SPECTA Processed Result for the Lafayette Area ................................................................. 90
3.15 Highway Map of Lafayette, Indiana .......................................................................................... 91
3.16 S, T, B, P, and D Transformations on $a$ .................................................................................. 94
3.17 $\beta = D_2/(P^{1.1/q}(S^{1/v}(a))$ .............................................................................................. 97
3.18 A Directed Graph. ..................................................................................................................... 97
3.19 The Sequence of Transformation of $a^*$ from $a$ .................................................................... 98
3.20 Pattern Primitives. .................................................................................................................... 104
3.21 Character E (a) and Its Tree Representation (b) ..................................................................... 104
3.22 $a^*$, the Binary Form of $a$ ....................................................................................................... 105
3.23 Distorted Character E (a), and Its Binary Tree Representation $a^{**}$ (b) ................................. 112
3.24 The Acceptance of $a^{*}$ by GECTA. .......................................................................................... 114
3.25 Input Format (a), Chain-Code Result (b), and Binary Tree Representation (c) ...................... 116
3.26 Sample Patterns of Character A, C, D, E, H. ............................................................................ 118
4.1  The Primitive Extraction of a Character P ............................................................................... 125
4.2  The Primitive Selection of a Line Segment .............................................................................. 126
4.3  The Primitive Extraction of a Character K .............................................................................. 128
Figure

4.4 The 51 Chain-Coded Character Patterns ........................................ 129
4.5 The Distance Between Similar and Dissimilar Patterns .................... 136
4.6 The Minimum Spanning Tree of the 51 Character Patterns .................. 138
4.7 The Result of Using K-Nearest Neighbor Recognition
   Rule - K=1, t=6. .......................................................................... 139
4.8 The Result of Using K-Nearest Neighbor Recognition
   Rule - K=3, t=6. .......................................................................... 140
4.9 The Result of Using K-Nearest Neighbor Recognition
   Rule - K=3, t=6.5. ....................................................................... 141
4.10 The Result of Classification According to Cluster Centers .............. 144
4.11 Flow Chart of the Clustering Algorithm 4.4 ................................. 151

5.1 Four Texture Patterns Obtained from Digitizing Pictures Found
   in Brodatz's Book, Textures ...................................................... 165
5.2 Two Tree Structures for Texture Modeling ..................................... 169
5.3 Pattern and its Tree Representation ............................................. 170
5.4 A Regular Texture Pattern ......................................................... 174
5.5 A Distorted Texture Pattern Which Can be Generated by G2 .......... 174
5.6 A Random Texture Pattern Which Could be Generated by G2 .......... 178
5.7 Binary Pictures of Patterns in Figure 5.1 ...................................... 179
5.8 Windowed Patterns of Hexagonal Tessellation ................................. 184
5.9 The Generation of a Hexagonal Pattern by Using Grammars
   G3 and G3' ............................................................................ 184
5.10 The Generated Regular Hexagonals ............................................. 188
5.11 The Generated Regular Hexagonals ............................................. 189
5.12 Simulated Result of Pattern D34 .................................................. 190
5.13 The Ideal Texture of Pattern D34 ................................................ 192
5.14 Basic Patterns of Figure 5.13 ....................................................... 193
Figure
5.15 Windowed Pattern Primitives. ....................... 193
5.16 Synthesis Results of Pattern D22 ................... 195
5.17 The Syntactic Generation of Water Waves ........... 197
5.18 Synthesis Results of Pattern D38 ................... 200
5.19 Synthesis Results of Pattern D68 ................... 202
5.20 Pictorial Data for Texture Discrimination ........... 203
5.21 Binary Picture of Figure 5.20 ...................... 204
5.22 Discrimination Result ............................... 207

Appendix Figure
C.1 Typical Patterns Generated From $G_H$ .................. 231
The problem of modeling, analysis and reconstruction of noisy and/or distorted syntactic patterns is studied. Segmentation errors and primitive extraction errors can be treated as syntax errors and defined in terms of language transformation rules. Three types of error transformations are defined on strings, namely substitution, insertion and deletion. Consequently, the parser constructed according to the grammar generating the strings and the three types of transformations is called the error-correcting parser. This technique is also extended to tree languages. In formulating error-correcting tree automata (ECTA), five types of error-transformations on trees are defined, namely, substitution, split, stretch, branch and deletion. By way of using language transformations, the distance between two sentences can be determined. A definition of distance between a sentence and a language is proposed. Based on this definition, a clustering procedure is proposed, where error-correcting parsers are employed to determine the distance between an input syntactic pattern and a formed cluster, or a language. Finally, using the error-correcting parsing techniques, real data examples on texture modeling and discrimination are presented.
1.1 Purpose

During the past decade, there has been an increasing interest in pattern recognition. Most of the developments in the theory and applications of pattern recognition use the statistical approach [1-3]. In order to represent the structural information contained in the patterns, the syntactic or structural approach has been proposed [4-5]. This approach draws an analogy between the structure of patterns and the syntax of languages. The precision of syntactic specification provides the recognition procedure not only the capability of classifying patterns but also the capacity of describing patterns. However, one of the weaknesses of this approach is the problem of recognizing noisy patterns. Several approaches have been used in dealing with noisy patterns, namely; stochastic grammars or discriminant grammars, sequential parsing or partial parsing methods, language transformations, and error-correcting parsers. The purpose of this research is to develop error-correcting parsing algorithms suitable for syntactic pattern recognition.

1.2 The Recognition of Noisy and Distorted Patterns

Using a syntactic approach to pattern recognition, a set of training patterns is first analyzed. A pattern, according to its structure, is divided into subpatterns. Subpatterns can be further divided into sub-subpatterns, and so on. The basic element is called a "pattern primitive."
Linguistic notations are used to describe a pattern in terms of primitives and relations between them as in a sentence. The set of sentences corresponding to the set of training patterns can be specified by a generative grammar called a "pattern grammar." Non-terminals in the pattern grammar represent the subpatterns and terminals represent primitives and possibly some relational symbols. The structure of patterns is characterized by production rules of the grammar.

In a recognition procedure, after preprocessing, segmentation and primitive extraction, an input pattern is represented as a sentence, then a parser is employed as a pattern recognizer. A parser is an algorithm based on a given pattern grammar, G, that can produce a complete syntactic description in the form of a parse tree of an input sentence if it belongs to L(G), the language generated from G. A block diagram of a syntactic pattern recognition system is given in Figure 1-1.

In a pattern classification problem, parsers are used to determine the membership of an input pattern. Grammars are constructed to characterize each of the classes of patterns. An input pattern is then parsed with respect to the pattern grammars one by one to decide which language (class) it belongs to.

In practical applications, there often exists pattern distortion and measurement noise causing segmentation and primitive extraction errors which ultimately result in a noisy representation (sentence), that is, it cannot be successfully analyzed by the parser. The following are situations that may cause the representation of a pattern to be noisy; (1) unpredictable distortions and variations, (2) simplified grammars.
Figure 1.1 Block diagram of a syntactic pattern recognition system.
(1) Unpredictable distortions and variations. Normally, one would like to construct a grammar which generates as much variety in patterns as possible. The construction of a grammar is based on a priori knowledge available; e.g. the given set of patterns, or predictable noise, distortion or variation of patterns. However, not all the distortions and variations of patterns are predictable. This uncertainty may cause some patterns to be rejected during recognition.

(2) Simplified grammars. In a classification problem, in order to avoid any ambiguity caused by the overlaps between languages, grammars may be constructed to exclude some known patterns as well as some expected distortions in order to be simpler and smaller. There is a decision which has to be made between the descriptive precision and the analysis efficiency of a grammar. One may construct a large grammar (with a large number of production rules) which generates a language that very closely yields the given set of patterns, or a simpler grammar which does not generate some of the known patterns but uses less parsing time and storage space.

Stochastic grammars have been suggested in resolving the uncertainty of patterns [6-8]. With a probability assigned to each production rule, the probability distribution of sentences generated from the stochastic grammar can be used to model the probability distribution of patterns. Normal patterns are discriminated from noisy patterns by their associated probabilities. This approach requires a large amount of training data in order to make a meaningful probability distribution of patterns.

Page and Filipski [9], propose the use of a discriminant grammar which is an extension of the stochastic grammar approach. The generated language from a discriminant grammar is supposed to include all the classes
of patterns under consideration. In a discriminant grammar, a number is associated with each production rule. By adding the numbers corresponding to each use of a production rule in a derivation of a sentence, a number associated with the sentence is derived. The language is partitioned into decision regions by comparing this number to a predetermined cut point. A discriminant grammar can also be used in making a Bayes' decision between two stochastic languages. In this case, the number assigned to each production rule is obtained from the probability distribution of sentences. Therefore, the construction of discriminant grammars faces a similar problem to that of stochastic grammars, namely, that a very large amount of training data is necessary in order to make a meaningful probability distribution.

An interesting feature that stochastic grammars and discriminant grammars have is the sequential parsing. Persoon and Fu [10] proposed an algorithm of sequential parsing for stochastic context-free grammars. Using a stopping rule, only part of the input string needs to be scanned when a decision is made. The classification rule used is Bayes' decision rule. Page and Filipski [9] also proposed a scheme for sequential parsing of discriminant grammars based on the sequential probability ratio test. Using the sequential parsing method, since only the left part of a string is involved in the decision making process, one may construct the pattern grammars in such a way that the most informative, distinctive subpatterns and primitives are generated first. Consequently, the sequential classification schemes demonstrate an error tolerance capability to some extent. The error tolerance of sequential parsing is investigated in Chapter 2 of this thesis.
Ali and Pavlidis [11] used a similar idea for the construction of parsers for handwritten numerals. Finite state grammars are used for the characterization of each class (numeral). A set of finite state automata is designed to read primitives considered to be discriminant, while unimportant primitives are neglected. Since the recognizer ignores some irrelevant details which are extracted by pattern description algorithms it can thus reduce recognition errors.

The use of language transformation for the representation of special types of pattern distortions, such as scaling, rotation and replacement, etc. was suggested by Fu and Bhargava [12]. This concept also appears in Aho and Peterson's paper [13] where error transformation for substitution, deletion and insertion errors are defined. Aho and Peterson further expand the original grammar to include error transformations as production rules. Based on the expanded grammar an error-correcting parsing algorithm was formulated for substitution, deletion and insertion errors in general. The correction satisfies the minimum-distance criterion.

The approach of using error-correcting parsers is the method used in this research. A parser constructed on a given grammar, \( G \), performs the function of analysing an input string, \( x \). The analysis result is a complete parse of \( x \), if \( x \) is in \( L(G) \), the language generated from \( G \). From the parse, a derivation tree which represents the structure of \( x \) can be reconstructed. Suppose that \( x \) is not in \( L(G) \). Then the parser can, at most, generate a partial parse. Therefore, for a given grammar, \( G \), a parser can be used to answer the membership problem, it can also be used to describe the syntax structure if the input sentence is in \( L(G) \) [14]. An error-correcting parser is designed to generate a complete parse even
if the input sentence is not in $L(G)$. Hence, using an error-correcting parser in a pattern recognition problem, a noisy pattern can be successfully analyzed and recognized. Block diagrams are given in Figures 1-2(a) and (b) to illustrate the function of a parser and an error-correcting parser. A corrected pattern may further be reconstructed from the parse. The use of error-correcting parsers in dealing with noisy patterns has the following two advantages over other methods.

1) Improvement of recognition performance under inadequate training. The construction of grammars, manually or automatic, is an important part in the design of a syntactic pattern recognition system. An elaborate design gives better recognition performance, but such a design is certainly data dependent. Inadequate inference procedures or insufficient training data will result in a poorly constructed grammar, and consequently poor recognition performance. The use of error-correcting parsers as a pattern recognizer will compensate this difficulty in grammar construction. Hence, even with a rather poorly constructed grammar, the use of error-correcting parsers can give satisfactory recognition results.

2) Correction of noisy patterns. When pattern grammars are constructed from noise-free patterns only, noisy or distorted patterns can be corrected by using error-correcting parsers.

1.3 Survey of Error-Correcting Parsing

The idea of using syntax rules in correcting program errors or punctuation errors arose with the design of syntax-directed compilers. Because the syntactic specifications are precise, syntax analysis not only plays a central role in the organization of compilers, it also provides error detection and recovery capability within the compiler.
Figure 1.2 Syntax analysis using a parser (a) and an error-correcting parser (b)

(a) Sentence $x$ parsed, if $x \in L(G)$
    Parsing fails, if $x \notin L(G)$

(b) Sentence $x$ parsed, if $z \in L(G)$
    $(z = x$ if $x \in L(G))$
Most error-recovery strategies take the point where parsing fails to continue as the point of detection of errors, [15-21]. A recovery action is then applied to suppress the error so that parsing can be resumed. Diagnostic information or corrections may also be generated at this point. In [15], Iron attempts to supply some recovery action at each point where an inconsistency is detected by a top-down parser. Part of the input sentence is replaced based on the context of the error and productions of the grammar to allow parsing to continue. Similar to Iron's idea, Gries [16] proposes more sophisticated error-recovery strategies for top-down parsers. Using a precedence grammar, an error is detected when no precedence relation exists between the incoming terminal and the symbol at the top of the stack, or the phrase to be reduced has no equivalent right hand side of a production. Wirth [17] proposes an error recovery algorithm by scanning tables of error rules for an entry which applies to the erroneous condition. The table of error rules is based on designers' knowledge of common programming errors and appropriate recovery actions. Leinrius [18] discusses strategies of isolating the smallest potential phrase and then makes the required replacement. Graham and Rhodes [19], suggest a weighted minimum distance measure for finding a "closest fit" local corrections when an error recovery routine faces numerous choice of next move.

Error-correcting for compilers emphasizes early detection of errors and generating accurate diagnosis. Nevertheless, error detection may arbitrarily be delayed. In a study by Aho and Ullman [23], they concluded that a precedence parser will not always detect an error as soon as a corresponding LR(1) parser. Repairing techniques for LR and LL parsers are still subjects to be studied.
An alternative to the heuristic approach for error-correcting parsing is the minimum-distance error-correcting techniques. All the potential errors and their corrections are recorded during parsing. After the entire string has been processed, a derivation that satisfies the minimum-distance criterion will be generated. Aho and Peterson formulate substitution, deletion and insertion errors in terms of error transformations [13]. Distance between two sentences is defined as the least number of transformations used to derive one from the other. The original grammar is then expanded to include error transformations in the set of production rules such that its generated language contains all the possible erroneous sentences. The parsing algorithm is a modified Earley's parser [34] with provisions added for the bookkeeping of the number of error transformations used. During parsing, the potential derivation that uses the least number of error production rules is placed in the parse list. By the time a parsing is completed, the minimum-distance correction is also achieved. Instead of formulating error production rules, Lyon proposes a scheme that puts all the possible corrections such as the substitutions, deletions and insertions of the currently scanned symbol in the parse list [24]. Setting limitations on the number of local errors and global errors are also suggested by Lyon to decrease the parsing time and memory storage.

Although the minimum-distance error-correcting parsers (MDECP) use a similarity criterion in their searching for the syntactically correct sentence they are considered impractical from compiler design point of view. Compiler designers' interests are in methods that generate accurate diagnostic information and continue parsing more than that of automatic correction, since part of the debugging and correction can be left to the programmer.
A spelling error correction technique is proposed by Morgan [25]. He uses a heuristic method to search for a good match of the input string from a table of code words. A more rigorous approach in finding a best match from a finite set of strings for an input string is to use the algorithm given by Wagner and Fisher [26]. Error-correcting parsers for regular language are proposed by Wagner [27] and Thomason [28] respectively.

The second interesting application of error-correcting parsing is in syntactic decoding systems where errors are caused from noisy communication channels [29-32]. In modeling the randomness of noisy channels, it is essential that the designed probabilistic model can be applied to the syntactic processing of linguistic information. Bahl and Jelinek [30], proposed a first order Markov chain model for noisy channels in which an input sequence, \( a_1 a_2 ... a_n \) can produce output sequence \( b_1 b_2 ... b_m \) of varying lengths. This is done by associating with each input symbol \( a_k \) a probabilistic finite state machine. A null transition, self-loop transitions and transitions producing output other than \( a_k \) are added to the finite state machine for modeling deletion, insertion and substitution errors. In Fung and Fu's probabilistic deformation model, context-free languages with substitution errors are considered [31].

Let \( x = a_1 a_2 ... a_n \) be a string, the error occurred on a symbol, \( a_k \) is assumed to be independent from its context in \( x \). Therefore, the probability that \( y = b_1 b_2 ... b_n \) is an error deformed string of \( x \) is defined as follows

\[
q(y|x) = \prod_{i=1}^{n} q(b_i|a_i)
\]

where \( q(b_i|a_i) \) is the probability that \( b_i \) substitutes for \( a_i \). Let \( L(G_1) \) and \( L(G_2) \) be two languages, the maximum-likelihood decision rule proposed by Fung and Fu is
Given a grammar $G$, the parser designed for the searching of $x$ that satisfies the maximum-likelihood criterion, 
\[
\max_{x \in L(G)} q(y|x) \geq \max_{x \in L(G_1)} q(y|x) \quad \text{decide} \quad y \in L(G_1) \\
\max_{x \in L(G_2)} q(y|x) \quad \text{decide} \quad y \in L(G_2)
\]

A modified Cocke-Younger-Kasami parser is used by Fung and Fu. Thompson [32] formulates error-correcting parsers for stochastic grammar in Greibach normal form (GNF). The approach of using expanded grammars in which substitution, deletion, insertion and transposition errors are added as error production rules is also used here. The four types of errors are treated separately. Four separate algorithms each of which copes with one type of error are then combined to correct simultaneous errors. Although Thompson gives no discussion on complexity of this ECP, he points out that the original top-down parser for GNF has already exponential growth in computational complexity. Ambiguity caused by the four expanded grammars is expected to increase this complexity. The combined algorithm would further compound time complexity with the enormity of the bookkeeping problem. It is suggested to combine the algorithm serially to simplify the process. However, the use of serial combination may not always give maximum-likelihood correction.

The idea of using error-correcting parsers in the syntactic recognition of noisy patterns has also been suggested [31,33]. Since pattern recognition systems handle a variety of types of input data, such as pictorial data [35-39], waveforms [40-43], speech patterns, program schemes [44-45], or data files [46] etc., the metalanguages used to describe patterns can be sets of strings, trees or graphs. In addition to type 1,
type 2, and type 3 grammars, programmed grammars [47-48], transition networks [49], tree grammars [50], graph grammars [45,51], web grammars [52,53], array grammars [54-56], and many others have been used for pattern analysis. An error-correcting parsing scheme for context-sensitive grammars is proposed by Tanaka and Fu [57].

Error-correcting parsing for syntactic pattern recognition is still at a beginning stage. Most existing pattern grammars do not have their corresponding error-correcting parsers. Similarity measure is a key point in designing such a parser for a pattern recognition system. The idea of using language transformations for the modeling of primitive extraction and segmentation errors, and constructing a parser based on the expanded grammar which includes transformation rules, provides a global distance measurement. We shall use this approach in formulating error-correcting parsers for stochastic and non-stochastic context-free languages and tree languages. A minimum-distance error-correcting parser for context-free program grammar is also presented.

1.4 Thesis Organization

In Chapter 2, the distance between two strings is measured in terms of the three defined transformations, namely, substitution, deletion and insertion transformations. This measurement provides a similarity measure between syntactic patterns. Error-correcting parsers are formulated based on a similarity criterion; e.g. the minimum-distance criterion. Definitions on distance between a string and a language are proposed. A minimum-distance classification system using a modified error-correcting parsing algorithm is presented. A similar approach is applied to a stochastic model where stochastic languages, deformation probabilities and maximum-likelihood criterion are used.
The problem of error-correcting syntax analysis for tree languages is studied in Chapter 3. Syntax errors on trees are defined in terms of five types of transformation, namely, substitution, deletion, stretch, split and branch. In the formulation of error-correcting tree automata (ECTA), transformations made on each terminal symbol are added to the automata in the form of transition functions. Two types of ECTA are proposed: one for substitution errors called SPECTA and one for all five types of errors called GECTA. Real data examples of using SPECTA for LANDSAT data interpretation and GECTA for character recognition are presented.

Chapter 4 and Chapter 5 describes two potential applications of error-correcting parsers in the area of pattern recognition. As the distance between a sentence (a syntactic pattern) and a language (a group of syntactic patterns) is defined, and its computation is implemented by using error-correcting parsers, a clustering procedure for syntactic patterns is proposed in Chapter 4. A character recognition experiment is given as an illustrative example.

A syntactic model for the generation and the discrimination of structured textures is described in Chapter 5. A texture pattern is first divided into fixed-sized windows. Windowed patterns belonging to the same class of texture are then characterized by a tree grammar. The uncertainty existing in texture patterns; e.g. local noise, structure distortion, makes them impossible to be fully characterized by the constructed grammars. Therefore, SPECTA's are used as texture discriminators. Texture patterns generated by tree grammars are illustrated. Discrimination results are also given.
A short summary of this research and suggestions on further work can be found in Chapter 6.
CHAPTER 2
ERROR-CORRECTING PARSING FOR STRING LANGUAGES

2.1 Introduction

In this chapter, a distance between two strings is first defined and then extended to the distance between a string and a language. The distance between two strings is defined in terms of the minimum number of error transformations used to derive one from the other by Aho and Peterson [13]. When the error transformations are defined in terms of substitution, deletion and insertion errors, the distance measurement coincides with the definition of Levenshtein metric [61]. In Section 2.2, error transformations are applied to weighted Levenshtein metric. Also, a new metric, simply called weighted metric, which would reflect the difference of the same type of error made on different terminals is proposed. This extension provides a similarity measure between two sentences more closely related to the similarity of their corresponding patterns.

For a given input string $y$ and a given grammar $G$, a minimum-distance error-correcting parser (MDECP) is an algorithm that searches for a sentence $z$ in $L(G)$ such that the distance between $z$ and $y$, $d(z,y)$ is the minimum among the distances between all the sentences in $L(G)$ and $y$. The algorithm also generates the value of $d(z,y)$. We simply define this value to be the distance between $L(G)$ and $y$ and denote it as $d_1(L(G),y)$.

When a given grammar is a context-free grammar (CFG), its MDECP can be implemented by modifying an Earley's parsing algorithm. We also extend
the definition of the distance between \( L(G) \) and \( y \), \( d_1(L(G),y) \), to the
definition of \( d_K(L(G),y) \), the average distance between \( y \) and the \( K \)
sentences in \( L(G) \) that are the nearest to \( y \). The computation of \( d_K(L(G),y) \)
can be implemented by further modification of the algorithm of MDECP. In
Section 2.2.3 a minimum-distance decision rule is proposed for classi-
fication of syntactic patterns.

An algorithm of MDECP for context-free programmed grammars (CFPG) is
given in Section 2.3. This algorithm is restricted to Levenshtein's
distance only. In pattern recognition, the context-free programmed grammars
are considered having higher descriptive power than the context-free
grammar. It is proved by Rosenkrantz, that the set of languages gener-
ated by CFPG's properly contains the set of context-free languages, and
is properly contained within the set of context-sensitive languages [60].
A CFPG generating a context-sensitive language may be selected in order
to describe the patterns effectively. Although a context-sensitive
grammar (CSG) may as well be used, the parsing based on a CFPG has better
analysis efficiency than a CSG.

In Section 2.4, the stochastic deformation model of substitution
errors proposed by Fung and Fu [31] is first extended to include deletion
and insertion errors. Based on the deformation model, the deformation
probabilities can be estimated from the observations of these errors.
Similar to the use of error transformations proposed in [13], the stochastic
defection model is introduced into the original stochastic context-free
grammar (SCFG). The Earley's parser is modified for the searching of
the most likely error-correction based on the maximum-likelihood criterion.
We shall call this algorithm the maximum-likelihood error-correcting parser
(MLECP). For an input sentence $y$ and a given SCFG, $G_s$, a MLECP generates the most likely correction of $y$, $x$. Let $p(x)$ be the probability of $x$ in $L(G_s)$. The MLECP also computes the value of $q(y|x)p(x)$ where $q(y|x)$ is the deformation probability of $y$ given $x$. We shall interpret the term, $q(y|x)p(x)$, as the deformation probability of $y$ given $L(G_s)$, denoted as $q(y|G_s^*)$. If the a priori probability of each grammar is known, Bayes' decision rule is proposed as a decision criterion.

Due to the inefficiency of such an error-correcting parser, we are also interested in the improvement of parsing speed. Persoon and Fu have proposed a sequential classification algorithm (SCA) for stochastic context-free languages [10]. The error-tolerance capability of SCA is investigated in Section 2.4.4. We further modify the SCA to classify noisy sentences using the error-correcting approach. Experimental results illustrate that within a tolerable percentage of misrecognition, the speed of SCA is faster than that of MLECP.

2.2 Minimum-Distance Error-Correcting Parsing for Context-Free Languages

Following the notations used in [14], the definition of grammars and languages is briefly reviewed.

**Definition 2.1.** A grammar is a 4-tuple $G = (N, \Sigma, P, S)$ where

1. $N$ is a finite set of nonterminal symbols
2. $\Sigma$ is a finite set of terminal symbols disjoint from $N$.
3. $P$ is a finite subset of

$$\{(\alpha, \beta)^*N(\alpha^*)^*|N(\alpha^*)^*\}$$

An element $(\alpha, \beta)$ in $P$ will be written $\alpha + \beta$ and called a production.
Definition 2.2. The language generated by a grammar $G$, denoted $L(G)$, is the set of sentences generated by $G$. Thus,

$$L(G) = \{ \omega | \omega \text{ is in } \Sigma^*, \text{ and } S \rightarrow^* \omega \}$$

where a relation $\rightarrow$ on $(\Sigma^*)^*$ is defined as follows: if $\alpha \beta \gamma$ is in $(\Sigma^*)^*$ and $\beta \rightarrow \delta$ is a production rule in $P$ then $\alpha \beta \gamma \rightarrow \alpha \delta \gamma$, and $\rightarrow^*$ denotes the reflexive and transitive closure of $\rightarrow$.

If each production in $P$ is of the form $A \rightarrow \alpha$, where $A$ is in $N$ and $\alpha$ is in $(\Sigma^*)^*$ then the grammar $G = (N, \Sigma, P, S)$ is classified as a context-free grammar. The set of languages that can be generated from context-free grammars are called context-free languages.

2.2.1 A Similarity Measure for Syntactic Patterns

In [13], errors in a string are considered to be the three types: substitution, deletion and insertion errors, and treated as syntax errors by defining transformations from $\Sigma^*$ to a subset of $\Sigma^*$.

Definition 2.3. For two strings, $x, y \in \Sigma^*$, we can define a transformation $T: \Sigma^* \rightarrow \Sigma^*$ such that $y \in T(x)$. The following three transformations are introduced:

1. Substitution error transformation

   $$\omega_1 \alpha \omega_2 \xrightarrow{T_S} \omega_1 \beta \omega_2, \text{ for all } a, b \in \Sigma, a \neq b,$$

2. Deletion error transformation

   $$\omega_1 \alpha \omega_2 \xrightarrow{T_D} \omega_1 \omega_2, \text{ for all } a \in \Sigma$$
(3) Insertion error transformation

\[ T_I: \omega_1 \omega_2 \rightarrow \omega_1 \omega \omega_2, \text{ for all } \omega \in \Sigma \]

where \( \omega_1, \omega_2 \in \Sigma^* \)

**Definition 2.4.** The distance between two strings \( x, y \in \Sigma^* \), \( d^L(x,y) \), is defined as the smallest number of transformations required to derive \( y \) from \( x \).

**Example 2.1.** Given a sentence \( x = \text{cbabdbbb} \) and a sentence \( y = \text{cbbabdbdb} \), then

\[
\begin{align*}
\text{x} & \rightarrow \text{cbabdbb} \\
\text{T_S} & \rightarrow \text{cbabbb} \\
\text{T_S} & \rightarrow \text{cbbabdb} \\
\text{T_I} & \rightarrow \text{cbbabdb} = \text{y}
\end{align*}
\]

The minimum number of transformations required to transform \( x \) to \( y \) is three, thus, \( d^L(x,y) = 3 \).

The metric defined in Definition 2.4 gives exactly the Levenshtein distance of two strings [61]. A weighted Levenshtein distance can be defined by assigning nonnegative numbers \( \alpha, \gamma \) and \( \delta \) to transformations \( T_S, T_D \) and \( T_I \) respectively. Let \( x, y \in \Sigma^* \) be two strings, and let \( J \) be a sequence of transformations used to derive \( y \) from \( x \), then the weighted Levenshtein distance between \( x \) and \( y \), denoted as \( d^w(x,y) \) is

\[
d^w(x,y) = \min \left\{ \alpha \cdot k_j + \gamma \cdot m_j + \delta \cdot n_j \right\}
\]

where \( k_j, m_j \) and \( n_j \) are the number of substitution, deletion, and insertion transformations respectively in \( J \).

We shall propose a weighted metric that would reflect the difference of the same type of error made on different terminals. Let the weights associated with error transformations on terminal \( a \) in a string \( \omega_1 \omega_2 \) where
a ∈ Σ, ω₁ and ω₂ ∈ Σ⁺, be defined as follows:

1. \( ω₁aω₂ \overset{T_S}{\longrightarrow} ω₁bω₂ \) for \( b ∈ Σ, b \neq a \), where \( S(a,b) \) is the cost of substituting \( a \) for \( b \). Let \( S(a,a) = 0 \).

2. \( ω₁aω₂ \overset{T_D}{\longrightarrow} ω₁ω₂ \) where \( D(a) \) is the cost of deleting \( a \) from \( ω₁aω₂ \).

3. \( ω₁aω₂ \overset{T_I}{\longrightarrow} ω₁b′aω₂ \) for \( b ∈ Σ \), where \( I(a,b) \) is the cost of inserting \( b \) in front of \( a \).

We further define the weight of inserting a terminal \( b \) at the end of a string \( x \) to be,

\[ \begin{align*}
\text{(4) } x & \overset{T_I'}{\longrightarrow} xb, \text{ for } b ∈ Σ. 
\end{align*} \]

Let \( x, y ∈ Σ^* \) be two strings, and \( J \) be a sequence of transformations used to derive \( y \) from \( x \). Let \( |J| \) be defined as the sum of the weights associated with transformations in \( J \), then the weighted distance between \( x \) and \( y \), \( d^W(x,y) \) is defined as

\[ d^W(x,y) = \min \{|J|\} \]  

Equation (2.2) can be illustrated by a graphical interpretation. From point B to point E, each path in the lattice shown in Figure 2.1 corresponds to a sequence of transformations used to derive \( y \) from \( x \). A horizontal branch indicates an insertion transformation, a vertical branch indicates a deletion transformation, and a diagonal branch indicates a substitution or a non-error transformation. The weight assigned to a particular type of transformation on a particular symbol in \( x \) is labeled at its corresponding branch. Let \( J \) be a path in the lattice, then \( |J| \) is the sum of weight associated with each branch in \( J \). The distance,
Figure 2.1 A graphic Interpretation of metric W
\(d^w(x,y)\), is the weight associated with the minimum-weight path.

We shall refer to the Levenshtein distance, weighted Levenshtein distance and weighted distance as metric \(\text{L, w, and W respectively, and use } d(x,y)\) to denote the distance between \(x\) and \(y\) based on any of the three metrics.

2.2.2 A Minimum-Distance Error-Correcting Parser

Let \(L(G)\) be a given language and \(y\) be a given sentence the essence of minimum-distance error-correcting parsing is to search for a sentence \(x\) in \(L(G)\) that satisfies the minimum distance criterion as follow

\[
d(x,y) = \min_{z \in L(G)} \{d(z,y)\}
\]  

We note that the minimum-distance correction of \(y\) is \(y\) itself if \(y \in L(G)\).

We shall extend the minimum-distance ECP proposed by Aho and Peterson [13] to all three types of metric; \(\text{L, w, and W.}\) In [13], the procedure for constructing an ECP starts with the modification of a given grammar \(G\) by adding the three types of error transformations in the form of production rules, called error productions. The grammar \(G\) is now expanded to \(G'\) such that \(L(G')\) includes not only \(L(G)\), but all possible sentences with the three types of errors. The parser constructed according to \(G'\) with a provision added to count the number of error productions used in a derivation is the error-correcting parser for \(G\). For a given sentence \(y\), the ECP will generate a parse, \(\Pi\), which consists of the smallest number of error productions. A sentence \(x\) in \(L(G)\) that satisfies the minimum-distance criterion (measured by using Levenshtein distance) can be generated from \(\Pi\) by eliminating error productions. With some modifications, this minimum-distance ECP can easily be extended to the three
metrics proposed in Section 2.2.1. We first give the algorithm of constructing an expanded grammar, in which the nonnegative numbers associated with error-productions are the weights associated with their corresponding error transformations with respect to the metric used.

Algorithm 2.1. Construction of expanded grammar

Input: A CFG \( G = (N, \Sigma, P, S) \)

Output: A CFG \( G' = (N', \Sigma', P', S') \) where \( P' \) is a set of weighted productions.

Method:

Step 1. \( N' = \{S'\} \cup \{E_a \mid a \in \Sigma\} \cup \Sigma' \cup \Sigma \)

Step 2. If \( A + a_0 b_1 a_1 b_2 \ldots b_m a_m \), \( m > 0 \) is a production in \( P \) such that \( a_1 \in \Sigma^* \) and \( b_1 \in \Sigma \), then add \( A + a_0 E_{b_1} a_1 E_{b_2} \ldots E_{b_m} a_m \) to \( P' \), where each \( E_{b_1} \) is a new non-terminal, \( E_{b_1} \in N' \) and \( 0 \) is the weight associated with this production.

Step 3. Add the following productions to \( P' \).

<table>
<thead>
<tr>
<th>Production Rule</th>
<th>Weight</th>
<th>( L )</th>
<th>( w )</th>
<th>( W ) (metric)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) ( S' + S )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>(b) ( S' + Sa )</td>
<td>1</td>
<td>( \delta )</td>
<td>1'(a)</td>
<td>for all ( a \in \Sigma' )</td>
</tr>
<tr>
<td>(c) ( E_a + a )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>for all ( a \in \Sigma )</td>
</tr>
<tr>
<td>(d) ( E_a + b )</td>
<td>1</td>
<td>( \gamma )</td>
<td>( S(a, b) )</td>
<td>for all ( a \in \Sigma, \ b \in \Sigma' ), and ( b \neq a )</td>
</tr>
<tr>
<td>(e) ( E_a + \lambda )</td>
<td>1</td>
<td>( \gamma )</td>
<td>( D(a) )</td>
<td>for all ( a \in \Sigma )</td>
</tr>
<tr>
<td>(f) ( E_a + bE_a )</td>
<td>1</td>
<td>( \delta )</td>
<td>1(a,b)</td>
<td>for all ( a \in \Sigma, \ b \in \Sigma' )</td>
</tr>
</tbody>
</table>
In Algorithm 2.1 the production rules added in Step 3(b), 3(d), 3(e) and 3(f) are called error productions. Each error production corresponds to one type of error transformation on a particular symbol in $\Sigma$. Therefore, the distance measured in terms of error transformations can be measured by error productions used in a derivation. The parser is a modified Earley's parsing algorithm with a provision added to accumulate the weights associated with productions used in a derivation. The algorithm is as follow.

Algorithm 2.2. Minimum-distance error-correcting parsing algorithm

Input: An expanded grammar $G' = (N', \Sigma', P', S')$ and an input string $y = b_1 b_2 ... b_m$ in $\Sigma^+$

Output: $l_0 l_1 ... l_m$ the parse list for $y$, and $d(x,y)$ where $x$ is the minimum-distance correction of $y$.

Method:

Step 1. Set $j = 0$. Then add $[E \rightarrow S', 0, 0]$ to $I_j$.

Step 2. If $[A \rightarrow a \cdot B B, i, \xi]$ is in $I_j$ and $B \rightarrow \gamma, \eta$ is a production rule in $P'$ then add item $[B \rightarrow \gamma, j, 0]$ to $I_j$.

Step 3. If $[A \rightarrow a \cdot i, \xi]$ is in $I_j$ and $[B \rightarrow \beta A \cdot \gamma, k, \zeta]$ is in $I_j$, and if no item of the form $[B \rightarrow \beta A \cdot \gamma, k, \phi]$ can be found in $I_j$, then add an item $[B \rightarrow \beta A \cdot \gamma, k, \eta + \xi + \zeta]$ to $I_j$ where $\zeta$ is the weight associated with production $A \rightarrow a$. If $[B \rightarrow \beta A \cdot \gamma, k, \phi]$ is already in $I_j$, then replace $\phi$ by $\eta + \xi + \zeta$ if $\phi > \eta + \xi + \zeta$.

Step 4. If $j = m$ go to Step 6, otherwise $j = j + 1$.

Step 5. For each item in $I_{j-1}$ of the form $[A \rightarrow a \cdot b_j \beta, i, \xi]$ add item $[A \rightarrow a b_j \cdot \beta, i, \xi]$ to $I_j$, go to Step 2.
Step 6. If item \([E \rightarrow S', 0, \hat{\xi}]\) is in \(I_m\). Then \(d(x, y) = \xi\), where \(x\) is the minimum-distance connection of \(y\), exit.

In Algorithm 2.2, the string \(x\), which is the minimum-distance correction of \(y\), can be derived from the parse of \(y\) by eliminating all the error productions. The extraction of the parse of \(y\) is the same as that described in Earley's algorithm.

2.2.3 A Minimum-Distance Classifier for Noisy Patterns

In Section 2.2.1, three metrics are proposed as similarity measures between two strings. We shall define the distance between a string and a given language based on any one of the three metrics as follows.

**Definition 2.5.** Let \(y\) be a sentence, and \(L(G)\) be a given language, the distance between \(L(G)\) and \(y\), \(d_K(L(G), y)\), where \(K\) is a given positive integer, is;

\[
d_K(L(G), y) = \min\left\{ \sum_{i=1}^{K} \frac{1}{K} d(z_i, y) | z_i \in L(G) \right\} \quad (2.4)
\]

In particular, if \(K = 1\), then

\[
d_1(L(G), y) = \min\{d(z, y) | z \in L(G)\} \quad (2.5)
\]

is the distance between \(y\) and its minimum-distance correction in \(L(G)\).

As the distance between a string (a syntactic pattern) and a language (a set of syntactic patterns) is defined, a minimum-distance decision rule can be stated as follows: suppose that there are two classes of patterns, \(C_1\) and \(C_2\) characterized by grammar \(G_1\) and \(G_2\) respectively. For a given syntactic pattern \(y\) with unknown classification,
Figure 2.2 A minimum-distance classifier for noisy patterns.
\[ C_1 \text{ decide } y \in C_2 \text{ if } d_K(L(G_1), y) < d_K(L(G_2), y) \tag{2.6} \]

A block diagram of a minimum-distance classification system is given in Figure 2.2.

For a given \( L(G) \) and a string \( y = b_1b_2 \ldots b_m \), the MDECP described in Algorithm 2.2 also generates the distance \( d_1(L(G), y) \). For the computation of \( d_K(L(G), y) \), \( K > 1 \), Algorithm 2.2 needs further modification. The following algorithm will generate parse lists in which each item in the list \( I_j \) is of the form \([A \rightarrow \alpha \cdot \beta, 1, (n_1, \pi_1), (n_2, \pi_2), \ldots, (n_e, \pi_e)]\) where \( e \leq K \). Each pair \((n_k, \pi_k)\), for \( 1 \leq k \leq e \), \( n_k \) is the weight associated with a derivation of substring \( b_{i+1} \ldots b_j \), and \( \pi_k \) is the corresponding correction of \( b_{i+1} \ldots b_j \) from this derivation.

**Algorithm 2.3.** Computation of \( d_K(L(G), y) \)

**Input:** An expanded grammar \( G' = (N', \Sigma', P', S') \), an input string \( y = b_1b_2 \ldots b_m \in \Sigma' \), and \( K \), a given positive integer.

**Output:** \( d_K(L(G), y) \) and \( x_1x_2 \ldots x_K \) the \( K \) nearest strings to \( y \) in \( L(G) \).

**Method:**

**Step 1.** Add item \([E \rightarrow \cdot S', 0, \phi] \) to \( l_0 \). Set \( j = 0 \).

**Step 2.** If \([A \rightarrow \alpha \cdot \beta, 1, (n_1, \pi_1), (n_2, \pi_2), \ldots, (n_e, \pi_e)] \) is in \( I_j \) and \( B \rightarrow \gamma \) is a production rule in \( P' \), then add item \([B \rightarrow \gamma, j, \phi] \) to \( I_j \).

**Step 3.** If \([A \rightarrow \alpha \cdot \beta, 1, (n_1, \pi_1), (n_2, \pi_2), \ldots, (n_e, \pi_e)] \) is in \( I_j \) and \([B \rightarrow \gamma, 1, (m_1, \tau_1), \ldots, (m_f, \tau_f)] \) is in \( I_j \),

(a) if item of the form \([A \rightarrow \alpha B \rightarrow \gamma, (l_{i1}, \eta_{i1}), (l_{i2}, \eta_{i2}), \ldots, (l_{if}, \eta_{if})] \) cannot be found in \( I_j \), then add \([A \rightarrow \alpha B \rightarrow \gamma, (l_{i1}, \eta_{i1}), (l_{i2}, \eta_{i2}), \ldots, (l_{i_g}, \eta_{i_g})] \) to \( I_j \), where each pair is chosen from the set \( N \).

\[ N = \{ (\xi + p \cdot m_q, \pi_p X) \mid 1 \leq p \leq e, 1 \leq q \leq f, \text{ and } X = B \} \]
B \rightarrow \beta, \xi \text{ is an error production, or } X = \tau_q, \text{ otherwise}, \text{ such that } i_1 \leq i_2 \leq \ldots \leq i_g \text{ are the } g \text{ smallest left hand side numbers having distinctive right hand side in } N.

g = |N|, \text{ if } |N| < K \text{ or } g = K \text{ if } |N| \geq K \text{ where } |N| \text{ is the number of elements in the set } N.

(b) if item of the form \([A \rightarrow aB \cdot \gamma, h, (i_1^1, n_1^1), \ldots (i_g^1, n_g^1)]\) is already in \(l_j\), then rearrange the set of pairs \((i_1^1, n_1^1), \ldots (i_g^1, n_g^1)\) to be \((i_1, n_1), \ldots (i_g, n_g)\) such that \(n_1, n_2, \ldots n_g\) are \(g\) distinctive strings and \(i_1 \leq i_2 \leq \ldots \leq i_g\) are \(g\) smallest number in the set \(N' = NU((i_1^1, n_1^1)), (i_2^1, n_2^1), \ldots (i_g^1, n_g^1)\) and \(g = |N'|\) if \(|N'| < K\), or \(g = K\) if \(|N'| \geq K\).

Step 4. Repeat Step 3 until no new item can be found. Then if \(j = m\), go to Step 6, otherwise \(j = j + 1\).

Step 5. For each item \([A \rightarrow aB \cdot \beta, l, (n_1, \pi_1), \ldots (n_e, \pi_e)]\) in \(l_j\), add \([A \rightarrow aB \cdot \beta, l, (n_1, \pi_1), \ldots (n_e, \pi_e)]\) to \(l_j\). Go to Step 2.

Step 6. If item \([E \rightarrow S' \rightarrow , 0, (n_1, \pi_1), \ldots (n_K, \pi_K)]\) is in \(l_m\), then

\[
d_k(L(G), y) = \sum_{i=1}^{K} \frac{1}{n_i}.
\]

The pairs \((n_i, \pi_i), 1 \leq i \leq K\) in the items of the parse lists are added for bookkeeping of the \(K\) nearest strings \(L(G)\) to the input \(y\). During the derivation of substring \(b_{j+1} \ldots b_j\), the corresponding corrected string are recorded such that identical strings caused by ambiguity of the grammar will not appear in the final set of the \(K\) nearest strings.
2.3. Error-Correcting Parsing for Context-Free Programmed Languages

2.3.1. Context-Free Programmed Grammar

Definition 2.6. A context-free programmed grammar (CFPG) is a 5-tuple \( G = (N, E, S, P, J) \) where

1. \( N \) is a finite set of non-terminals
2. \( E \) is a finite set of terminals
3. \( S \) is the start symbol in \( N \)
4. \( P \) is a finite set of programmed productions
5. \( J \) is a finite set of production lables

Each production in \( P \) consists of a label \( r \in J \), a core production of the form \( A \rightarrow \alpha \) where \( A \in N \), \( \alpha \in (N \cup E)^* \), and a success branch field and a failure branch field each consisting of elements from \( J \).

A derivation or generation in \( G \) proceeds as follows: the first production is applied to the start symbol \( S \); therefore, if production \( r \) is applied to the current sentential form \( \gamma \) to rewrite a nonterminal \( A \), and if \( \gamma \) contains at least one occurrence of \( A \), then the leftmost \( A \) is rewritten by the core of production \( r \) and the next production label is selected from the success branch field of \( r \); if the current sentential form does not contain \( x \), then the core of production \( r \) cannot be used and the next production label is selected from the failure branch field of \( r \); if the applicable branch field is empty, the derivation halts.

Example 2.2. Consider the CFPG \( G_p = (N, E, A, P, J) \) where

\[ N = \{A, B, C\}, E = \{a, b, c\}, J = \{1, 2, 3, 4, 5\} \] and \( P \):
A syntax analysis procedure for CFPG's has been proposed by Swain and Fu [48]. The algorithm can be explained by using the following example.

Example 2.3. Figure 2.3 is a schematic diagram of the analysis of the string abc with respect to grammar $G_p$. The notations used are explained as follows:

1. For any nonterminal $A \in N$, $A_i(y)A_i$ indicates that the string $y$ was generated from the nonterminal $A$ which was rewritten as $y$ at the $i$th step.

2. A downward arrow indicates a generative step. An upward arrow indicates backtracking. Backtracking is occurred when a generated sentential form is incompatible with input string. That is, the parsing detects at this step that if the analysis continues along the current path (or derivation), the string generated will be different from the input.

3. The branch labels have the form $\xi_k(\ell) = t$, where $\xi$ is $S$ (success) or $F$ (failure); subscript $k$ indicates the application of the $k$th production; $\ell$ indicates the selection of the $\ell$th branch in the
Figure 2.3 Analysis of abc with respect to $G_p$. 

\[ S_0(1) = 1 \]
\[ N_1(aBC)N_1 \]

\[ S_1(1) = 2 \]
\[ S_1(2) = 4 \]
\[ S_1(3) \text{ undefined} \]
\[ N_1(aN_2(bb)N_2C)N_1 \]
\[ N_1(aN_2(b)N_2C)N_1 \]
\[ \text{Terminate analysis of abc} \]

\[ F_4(1) = 5 \]
\[ F_4(2) \text{ undefined} \]
\[ N_1(aN_2(b)N_2N_3(c)N_3)N_1 \]

\[ F_5(1) = 1 \]
\[ \text{Successful analysis, report and backtrack} \]
ξ field, which is the branch to the production labeled t. By definition, \( \xi_k(\xi) = \emptyset \) indicates the termination of a path down the tree, which constitutes a successful parsing if the current sentential form is identical to the string being analyzed. The analysis of the string is complete when \( \xi_k(\xi) \) is undefined for \( k = 1 \) and some \( \xi \).

2.3.2 Error-Correcting Parsing Algorithms

Using the parsing algorithm for CFPG proposed by Swain and Fu, a parsing is considered to be a failure when all the paths in the analysis procedure terminates with a sentential form that is incompatible with the input string.

Example 2.4. Given a sentence, aabc, the parsing with respect to \( G \) is illustrated in Figure 2.4. In Figure 2.4, the analysis is complete without generating a successful parse. It concludes that aabc \( \notin L(G) \).

Let \( y \) be an input string, \( n \) be a preset positive integer. Suppose that a parsing is allowed to continue along the path in which an incompatible sentential form is generated. Backtracking is occurred only when a generated sentential form is considered to have the potential of generating a string with at least a distance \( n \) from the input string. Using this method, a syntactically correct sentence \( x \) can be generated such that \( d^L(x,y) \leq n \). A complete parsing procedure starts with \( n=0 \). If the analysis based on \( n \) fails, then \( n \) is increased by one and the analysis procedure is repeated, otherwise parsing is completed and \( d^L(x,y) = n \). The algorithm is described as follows.
Figure 2.4 The syntax analysis of aabc with respect to $G_p$. 

$S_0(1) = 1$

$S_1(1) = 2$

$S_1(2) = 4$

$S_1(3)$ undefined

terminate analysis, aabc is rejected.

$S_2(1) = 3$

$S_2(2) = 4$

$S_3(1) = 2$

$S_3(2) = 4$

$S_4(1) = 4$

$A_1(aB_2(aBB)B_2C)A_1$

$A_1(aB_2(b)B_2C)A_1$

$A_1(aB_2(aBB)B_2C_3(CC)C_3)A_1$

$A_1(aB_2(aBB)B_2C_3(CC)C_3)A_1$
Algorithm 2.4. MDECP for CFPG

Input: A CFPG $G_p = (N, \Sigma, S, P, J)$ and an input string $y = b_1b_2 \ldots b_m$.

Output: A string $x \in L(G_p)$, and $d^L(x, y)$

Method:

Step 1. Set $n=0$.

Step 2. Call subroutine PARSE($n$)

Step 3. If "Analysis Fail", $n=n+1$ go to Step 2.

Step 4. $d^L(x, y) = n$, exit.

Subroutine PARSE($n$)

Index and array:

$\xi_k(\ell)$ as explained in Example 2.3,

$f_k$ the left hand side of the $k$th rule,

$r_k$ the right hand side of the $k$th rule,

$X$ generated sentential form,

STACK($i, 1$) indicates $S$ or $F$, STACK($i, 2$) indicates the production
rule label. STACK($i, 3$) indicates the branch in $S$ or $F$ field of
the $i$th step in a derivation.

operator:

$X = X(f_t \leftarrow^i r_t)$ denotes that the left most nonterminal $f_t$ in $X$ is
replaced by $r_t$ in the $i$th step.

$X = X(f_t \rightarrow^i r_t)$ denotes that the substring $r_t$ in $X$ is replaced by
$f_t$, where $r_t$ is placed in $X$ at the $i$th step.

Method:

Step 1. Set $k=1$, $l=0$, $\ell=1$, $\xi=S$ and $X=r_k$.

Step 2. If $\xi_k(\ell)$ is undefined then go to Step 5, otherwise let

$t=\xi_k(\ell)$. If $\xi=F$, then go to Step 4.

Step 3. If $f_t$ cannot be found in $X$, then $\xi=F$, $\ell=1$ and $t=F_k(\ell)$. 
Step 4. Let i=1, STACK(1,1)=\$1, STACK(1, 2)=k, STACK(1, 3)=$. Let X = X(f_t \rightarrow r_t). Call subroutine COMPAT(n, X). If "Compatible," and if X is a terminal string, then "Analysis Success" Exit. otherwise k=t, \$=1, go to Step 2. If "Incompatible" go to Step 5.

Step 5. If i=0 then "Analysis Fail", exit. Otherwise X = X(f_t \rightarrow r_t), \$=STACK(1, 1) k=STACK(1, 2), \$=STACK(1, 3)+1, i=i-1 go to Step 2.

Subroutine COMPAT(n, X)

Index and array;
M = |X|, the length of sentential form X. Assume that X = B_1B_2...B_M.
D(i, j), stores the number of potential errors between B_1B_2...B_i and b_1b_2...b_j.

Method:

Step 1. D(0, 0)=0

Step 2. Do i=1 to M
D(1, 0) = D(1-1, 0)+1 if B_i is a terminal
D(1, 0) = D(1-1, 0) if B_i is a nonterminal

Step 3. Do j=1 to m
D(0, j) = D(0, j-1)+1

Step 4. Do i=1 to M, Do j=1 to m
(a) If B_i is a terminal and B_i \neq b_j then m_1 = D(i-1, j-1)+1 otherwise m_1 = D(j-1, j-1).
(b) If B_i is a terminal then m_2 = D(i-1, j)+1 otherwise m_2 = D(i-1, j).
(c) If $B_j$ is a terminal then $m_3 = D(1, j-1)+1$ otherwise $m_3 = D(1, j-1)$.

$$D(1, j) = \min (m_1, m_2, m_3)$$

Step 5. If $D(M, m) > n$ then it is "Incompatible" otherwise "Compatible". Exit.

Example 2.5. Let the string $aabc$ be parsed by Algorithm 2.4. The result of the first analysis ($n=0$) is shown in Figure 2.4. Since "Analysis Fail" is reported, the algorithm then increases $n$ by one. Figure 2.5 describes the result of the second analysis where an "Analysis Success" is reported. The algorithm generates the corrected string "abc", $d^*$(abc, aabc)=1.

2.4. Error-Correcting Parsing on a Stochastic Model

Basic notations and definitions of stochastic context-free grammar (SCFG) given in [62-63] are briefly reviewed.

Definition 2.7. A SCFG is a 4-tuples $G = (N, \Sigma, P, S)$ where,

- $N$ is a finite set of non-terminals,
- $\Sigma$ is a finite set of terminals,
- $P$ is a finite set of stochastic productions, each of which is of the form:

$$A_i \xrightarrow{P_{ij}} a_{ij}, \quad j = 1, 2, \ldots, n_i, \quad i = 1, 2, \ldots, \xi$$

where $n_i$ is the number of productions with $A_i$ at left-hand side, $\xi$ is the number of non-terminals, $A_i \in N$, $a_{ij} \in (\Sigma \cup \{\epsilon\})^*$, and $P_{ij}$ is the probability associated with this production. Furthermore,

$$0 < P_{ij} \leq 1 \text{ and } \sum_{j=1}^{n_i} P_{ij} = 1$$
Figure 2.5 The analysis of aabc when n=1
**Definition 2.8.** The stochastic context-free language (SCFL) generated by SCFG $G_s$ is

$$L(G_s) = \{(x, p(x)) \mid x \in \Sigma^*, S \xrightarrow{p_j} x, \quad j = 1, 2, \ldots, k, \text{ and } p(x) = \sum_{j=1}^{k} p_j\}$$

where $k$ is the number of all different derivations of $x$ from $S$, and $p_j$ is the probability associated with the $j$th distinctive derivation of $x$.

**Definition 2.9.** A SCFL is consistent if $\sum_{x \in L} p(x) = 1$.

### 2.4.1. A Stochastic Model for Syntax Errors

Following the notation of error transformations, the deformation probabilities associated with the three types of transformations, namely, $T_s$, $T_1$, $T_D$ are defined as follows:

1. $\omega_1 \omega_2 \xrightarrow{T_s, q_s(b|a)} \omega_1 b \omega_2$, where $q_s(b|a)$ is the probability of substituting terminal $a$ by $b$, and $a \neq b$.
2. $\omega_1 \omega_2 \xrightarrow{T_1, q_1(b|a)} \omega_1 b a \omega_2$, where $q_1(b|a)$ is the probability of inserting terminal $b$ in front of $a$.
3. $\omega_1 \omega_2 \xrightarrow{T_D, q_D(a)} \omega_1 \omega_2$, where $q_D(a)$ is the probability of deleting terminal $a$ from a string.
4. $x \xrightarrow{T_1, q_1(a)} x a$, where $q_1(a)$ is the probability of inserting terminal $a$ at the end of a string.

Let $q_s(a|a)$ be the probability that no error occurs on terminal $a$, which could be interpreted as the probability of the non-error transformation on $a$. Assume that for each terminal symbol there can be at
most one error existing. The deformation probabilities on this single-error model is consistent if

\[
\sum_{b \in \Sigma} q_s(b|a) + q_d(a) + \sum_{b \in \Sigma} q_1(b|a) = 1
\]

(2.7)

for all \(a \in \Sigma\)

Let \(a \in \Sigma^*\) be a substring the probability that symbol \(a\) is deformed to \(a\), denoted \(q(a|a)\), is defined as follows:

\[
q(a|a) = \begin{cases} 
q_d(a) & \text{if } a = \lambda \\
\max\{q_s(b|a), q_1(b|a)q_d(a)\} & \text{if } a = b \\
q_1(b_1|a) \cdots q_1(b_{\ell-1}|a) \max\{q_s(b_{\ell}|a), q_1(b_{\ell}|a)q_d(a)\} & \text{if } a = b_1 \cdots b_{\ell}, \ell \geq 1
\end{cases}
\]

(2.8)

Note that a substitution error can also be considered as an insertion transformation followed by a deletion transformation. The consistency of this multiple-error stochastic model defined in (2.8) can easily be proved from (2.7). Therefore, we have

\[
\sum_{a \in \Sigma^*} q(a|a) = 1
\]

(2.9)

The proof of (2.9) is given in Appendix A.

The probability of inserting \(a\), \(a \in \Sigma^*\), at the end of a string is defined as

\[
q'(a) = \begin{cases} 
1 - q'_1 & \text{when } a = \lambda \\
(1 - q'_1) q'_1(b_1) q'_1(b_2) \cdots q'_1(b_{\ell}) & \text{when } a = b_1 \cdots b_{\ell}, \ell \geq 1
\end{cases}
\]

(2.10)
where \( q'_1 = \sum_{a \in \Sigma} q'_1(a) \).

Furthermore,

\[
\sum_{a \in \Sigma} q'(a) = (1 - q'_1) \left\{ \sum_{l=0}^{\infty} (q'_1)^l \right\} = 1
\]  

(2.11)

It is also assumed that for any string of symbols \( a_1, a_2, \ldots, a_n \in \Sigma \), and strings \( a_1, a_2, \ldots, a_n, a_{n+1} \in \Sigma^* \) we have

\[
q(a_1 a_2 \ldots a_{n+1} | a_1 a_2 \ldots a_n) = q(a_1 | a_1) \ldots q(a_n | a_n) q'(a_{n+1})
\]  

(2.12)

With the deformation probability defined on each terminal of a string, the probability of deforming string \( x \) to string \( y \), \( q(y|x) \), where \( x = a_1 a_2 \ldots a_n \) is defined as

\[
q(y|x) = \max[ \prod_{l=1}^{n} q(a^l_j | a_j)] q'(a^l_{n+1})
\]  

(2.13)

where \( a_1 \ldots a^l_{n+1}, |a^l_j| \geq 0 \), is a partition of \( y \) into \( n+1 \) substrings, and

\( 1 \leq l \leq r \), \( r \) is the number of different ways of partitioning \( y \) into \( n+1 \) substrings.

A graphical interpretation of (2.12) is particularly illustrative. Consider an example, in which \( x = a_1 a_2 a_3 \) and \( y = b_1 b_2 b_3 b_4 \), whose lattice is shown in Figure 2.6. In this lattice, a horizontal branch indicates an insertion transformation, a vertical branch indicates a deletion transformation, and a diagonal branch indicates a substitution transformation or a non-error transformation.

In Figure 2.6, each traverse from point B to point E represents one way of deforming \( x \) into \( y \). The heavy line indicates that \( b_1 \) is an insertion in front of \( a_1 \), \( b_2 \) is a substitution for \( a_1, a_2 \) is deleted,
Figure 2.6 The stochastic deformation model described by a lattice.
b_3 substitutes a_3, and b_4 is inserted at the end. This deformation is made possible by partitioning \( y \) into \( a_1 a_2 a_3 a_4 \), where \( a_1 = b_1 b_2 \), \( a_2 = \lambda \), \( a_3 = b_3 \), \( a_4 = b_4 \). We have,

\[
q(a_1 a_2 a_3 a_4 | a_1 a_2 a_3) = q_1(b_1 | a_1) q_5(b_1 | a_1) \times q_6(a_2) \times q_5(b_3 | a_3) \times (1-q_1') q_1'(b_4)
\]

There are numerous ways of deforming \( x \) to \( y \), the deformation probability \( q(y|x) \) can be interpreted as the probability associated with the most likely path from point \( B \) to point \( E \).

2.4.2. Maximum-Likelihood Error-Correcting Parser (MLECP)

Let \( L(G_s) \) be a given SCFL, and \( y \) be a noisy string, \( y \notin L(G_s) \). The proposed maximum-likelihood error-correcting parsing algorithm is to search for a string \( x \), \( x \in L(G_s) \) such that

\[
q(y|x) p(x) = \max \{ q(y|z) p(z) | z \in L(G_s) \}
\]

(2.14)

where \( p(z) \) is the probability associated with \( z \) in \( L(G_s) \).

Similar to Algorithm 2.1, the construction of expanded grammars based on a stochastic deformation model is given as follows:

Algorithm 2.5. Construction of stochastic expanded grammar.

Input: A SCFG \( G_s = (N,E,P_s,S) \)

Output: \( G'_s = (N',\Sigma',P'_s,S') \) the stochastic expanded grammar.

Method:

Step 1. \( N' = N \cup \{ S' \} \cup \{ E_a | a \in \Sigma \} \).

Step 2. \( \Sigma' \supset \Sigma \)

Step 3. If \( A \rightarrow P a_1 b_1 a_2 b_2 a_3 \ldots b_m b_m' a \) is a production in \( P_s \), such that \( a_1 \) is in \( N^* \), and \( b_i \) is in \( \Sigma \), then add the
production $A \xrightarrow{p} a_0 E_{b_1} a_1 E_{b_2} \ldots E_{b_m} a_m$ to $P_s'$, where each $E_{b_i}$ is a new non-terminal, $E_{b_i} \in N'$.

**Step 4.** Add to $P_s'$ the productions

(a) $S' \xrightarrow{p_1} S$ where $p_1 = \sum_{a \in \Sigma} q_1'(a)$,

(b) $S' \xrightarrow{p_1'(a)} S'a$ for all $a \in \Sigma'$.

**Step 5.** For all $a \in \Sigma$, add to $P_s'$ the productions

(a) $E_a \xrightarrow{q_0(a|a)} a$,

(b) $E_a \xrightarrow{q_0(a|a)} b$ for all $b \in \Sigma'$, $b \neq a$,

(c) $E_a \xrightarrow{q_0(a|a)} \lambda$,

(d) $E_a \xrightarrow{q_1'(b|a)} b E_a$ for all $b \in \Sigma'$.

Suppose that $y$ is an error-deformed string of $x$, $x = a_1 a_2 \ldots a_n$. By using productions added to $P_s'$ by Step 3, we have $S \xrightarrow{p_1 G_s} X$, where $X = E_{a_1} E_{a_2} \ldots E_{a_n}$ if and only if $S \xrightarrow{G_s} x$, where $p_1$ is the $l$th derivation of $x$ in $G_s$. Applying Step 4(a) first and then repeatedly applying Step 4(b), we can further derive $S' \xrightarrow{G_s} X a_{n+1}$ where $p_1' = p_1 q_1'(a_{n+1})$.

The productions in Step 5 generate $E_{a_i} \xrightarrow{G_s} a_i$ for all $1 \leq i \leq n$, if $a_1, a_2, \ldots, a_{n+1}$ is a partition of $y$. Step 5(a), (b), (c), and (d) correspond to non-error transformation, substitution transformation, deletion transformation, and insertion transformation which allows multiple insertions, respectively.

Thus, the stochastic language generated by $G_s$ is

$$L(G_s') = \{(y, p(y)) | y \in \Sigma^*, p(y) = \sum_{x \in G_s} \sum_{i=1}^r q_1'(y|x) p(x)\}$$
where \( r \) is the number of distinctive sequence of transformations to derive \( y \) from \( x \), and \( q^i(y|x) \) is the probability associated with the \( i \)th sequence, \( 1 \leq i \leq r \).

The consistency of \( L(G'_s) \) can be proved from equation (2.9), (2.11) and (2.12).

It is proposed to use a modified Earley's parser on \( G'_s \) to implement the searching of the most likely correction of a noisy input. The algorithm is essentially Earley's Algorithm with a provision added to keep accumulating the probabilities associated with each step of derivations.

Algorithm 2.6. Maximum-Likelihood Error-Correcting Algorithm

**Input:** A stochastic expanded grammar \( G'_s = (V'_f, \Sigma', P'_s, S) \) of \( G'_s \), and string \( y = b_1b_2\ldots b_m \) in \( \Sigma^* \).

**Output:** Parse lists of \( y \).

**Method:**

**Step 1.** Set \( j = 0 \), and add \([E \cdot S', 0, 1]\) to \( I_j \).

**Step 2.**

(a) If \([A + \alpha \cdot B\beta, i, p]\) is in \( I_j \), and \( B \xrightarrow{q} \gamma \) is a production in \( P'_s \) add item \([B + \gamma, j, 1]\) to \( I_j \).

(b) If \([A + \alpha \cdot i, p]\) is in \( I_j \) and \([B + \beta \cdot A\gamma, k, q]\) is in \( I_j \) and if no item of the form \([B + BA \cdot \gamma, k, r]\) can be found in \( I_j \), add a new item \([B + BA \cdot \gamma, k, tpq]\) to \( I_j \), where \( t \) is the probability associated with \( A + \alpha \) in \( P'_s \). If \([B + BA \cdot \gamma, k, r]\) is already in \( I_j \), then replace \( r \) by \( tpq \) if \( tpq > r \).
Step 3. If \( j = m \), go to Step 5. Otherwise, \( j = j+1 \).

Step 4. For each item in \( I_{j-1} \) of the form \([A \rightarrow a \cdot b_j \beta, 1, p]\)
add item \([A \rightarrow ab_j \cdot \beta, 1, p]\) to \( I_j \), go to Step 2.

Step 5. If item \([E \rightarrow S^1 \cdot \cdot, 0, p]\) is in \( I_m \), exit.

The extraction of the most likely correction of \( y, x \), that satisfies
equation (2.14) are the same as that in MDECP.

From Step 5, the probability, \( p \), in the item \([E \rightarrow S^1 \cdot \cdot, 0, p]\) of
the last parse list, \( I_m \), gives the value of \( q(y|x)p(x) \) for some \( x \in L(G_5) \)
that satisfies equation (2.14) if the stochastic grammar, \( G_5 \) is
unambiguous. Since whenever a substring has more than one derivation,
the algorithm chooses the one associated with the largest probability.
Consequently, the derived number, \( p \), is \( q(y|x)p_j(x) \), where \( x \) is the most
likely correction of \( y \) and \( p_j(x) \) is the probability associated with the
jth distinctive derivation of \( x \) with respect to \( G_5 \) (refer to Definition 2.8).
Therefore, only when \( x \) has one derivation can \( p \) be interpreted as
\( q(y|x)p(x) \).

2.4.3. Bayes Classification of Noisy Patterns

Assume that there are two classes of syntactic patterns, \( C_1 \) and \( C_2 \).
Let \( x \) be a pattern. Suppose that the probability density function for
\( x \) in \( C_1 \), \( p(x|C_1) \) and the a priori probability of \( C_1 \), \( P(C_1) \), where \( i = 1, 2 \),
are known. Using Bayes rule, the posteriori probability that \( x \) is in
class \( j \) is

\[
P(C_j|x) = \frac{p(x|C_j)P(C_j)}{\sum_{i=1}^{2} p(x|C_i)P(C_i)}, \text{ for } j = 1, 2 \tag{2.15}
\]
Then, the maximum-likelihood decision rule (or Bayes decision rule) is,

\[
\begin{align*}
    \text{decide } x \in & C_1 \text{ if } P(C_1 | x) \geq P(C_2 | x) \quad (2.16) \\
    \text{decide } x \in & C_2 \text{ if } P(C_1 | x) < P(C_2 | x)
\end{align*}
\]

Let \( \tilde{C}_1, \tilde{C}_2 \) be two sets of training patterns for \( C_1 \) and \( C_2 \) respectively. Two stochastic grammars \( G_1 = (N_1, \Sigma_1, P_1, S_1) \) and \( G_2 = (N_2, \Sigma_2, P_2, S_2) \) are constructed to characterize \( \tilde{C}_1 \) and \( \tilde{C}_2 \) respectively, such that the probability of a sentence in \( L(G_i) \) yields the probability of its corresponding syntactic pattern in \( \tilde{C}_i \), for \( i = 1, 2 \). Let the probability of \( x \) in \( L(G_i) \) be denoted as \( p(x | G_i) \), where \( x \) is a given sentence. \( p(x | G_i) \) can be taken as \( p(x | C_i) \) in equation (2.15). The maximum-likelihood decision rule is then applied for recognition, provided that \( x \) is in \( L(G_1) \cup L(G_2) \), [64]. We shall rewrite equation (2.15) and (2.16) as follow:

\[
\begin{align*}
    \text{decide } x \in & C_1 \text{ if } p(x | G_1)P(C_1) \geq p(x | G_2)P(C_2) \quad (2.17) \\
    \text{decide } x \in & C_2 \text{ if } p(x | G_1)P(C_1) < p(x | G_2)P(C_2)
\end{align*}
\]

Note that \( p(x | G_1) = 0 \) if \( x \not\in L(G_1) \).

The purpose of this section is to provide a recognition rule that minimize error rate for a given string, even if it is not in any of the languages under consideration. Let the deformation probabilities of terminals in \( G_1 \) and \( G_2 \) be known. Let \( G_1' \) and \( G_2' \) be the stochastic expanded grammars for \( G_1 \) and \( G_2 \) respectively according to the deformation probabilities. Given a string \( y \), we shall interpret the term \( q_1(y | x)p(x) \) that satisfies equation (2.14) as the probability that \( y \) is an error deformed string of \( L(G_1) \), and denote it as \( q(y | G_1') \) where \( q_1('''|''') \) denotes the deformation probability of terminals in \( G_1 \). Then equation (2.14) can be rewritten as
\[ q(y|G_1^{'}) = \max_{z} \{ q_1(y|z)P(z) | z \in L(G_1) \} \] (2.18)

The use of expanded grammar, \( G_1' \), enlarges the probability space from \( L(G_1) \) to \( L(G_1') \) in which the deformation probabilities are employed to adjust the density function \( p(x|G_1) \) for \( x \in L(G_1) \). Consequently, the probability density function defined in equation (2.18) for sentences in \( L(G_1') \) more closely yields the distribution of syntactic patterns in \( C_1 \). Let the recognition rule be as follows:

\[ \text{decide } y \in C_1 \text{ if } q_1(y|G_1')P(C_1) < q_2(y|G_2')P(C_2) \] (2.19)

2.4.4. An Illustrative Example

A chromosome pattern classification problem is used as an example. Consider that there are four different types of chromosomes-submedian, median, acrocentric, and telocentric—denoted as \( C_s \), \( C_m \), \( C_a \), and \( C_t \), respectively [64, 65]. The typical chromosome patterns of each of the four types are illustrated in Figure 2.8. Let the SCFG's for \( C_s \), \( C_m \), \( C_a \), and \( C_t \) be \( G_s \), \( G_m \), \( G_a \), and \( G_t \), respectively. Segmentation errors due to noise and distortion in input patterns are considered as insertion or deletion errors, and, primitive extraction errors are considered as substitution errors. With a set of training samples, we can estimate deformation probabilities based on the segmentation and primitive extraction errors committed by the syntactic pattern recognition system. Then the MLECP's are constructed from the estimated deformation probabilities and SCFG's for each type of chromosomes. A set of samples, generated from the stochastic expanded grammars \( G_s' \), \( G_m' \), \( G_a' \), and \( G_t' \) (Algorithm 2.5), is then used as the test samples. They are classified by a Bayes
Figure 2.7 Bayes classification system for noisy strings
Figure 2.8 The typical chromosome patterns and their string representations.
classification system shown in Figure 2.7. All the programs in this paper are written in Fortran IV on a CDC 6500 computer.

(A) Pattern Grammars

Let the apriori probabilities \( P(C_S), P(C_M), P(C_A), \) and \( P(C_T) \) be \( .5, .26, .20, \) and \( .04 \), respectively [64]. The pattern grammars \( G_S, G_M, G_A, \) and \( G_T \) are given in Table 2.1. The deformation probabilities of each terminal are given in Table 2.2. For illustration only, we assume that the deformation probabilities of different classes are the same, the probabilities in Table 2.1 and Table 2.2 are rather arbitrarily assigned.

\[ G_S = (N_S, \Sigma_S, P_S, S) \]

\[ G_M = (N_M, \Sigma_M, P_M, S) \]

\[ P_S: \]

- \( S \rightarrow .5 \rightarrow WU \)
- \( S \rightarrow .5 \rightarrow UW \)
- \( W \rightarrow 1 \rightarrow cM \)
- \( U \rightarrow 1 \rightarrow cN \)
- \( M \rightarrow 1 \rightarrow bRb \)
- \( N \rightarrow 1 \rightarrow bLb \)
- \( R \rightarrow .55 \rightarrow bRb \)
- \( R \rightarrow .45 \rightarrow aQF \)
- \( Q \rightarrow .6 \rightarrow bQb \)
- \( Q \rightarrow .4 \rightarrow bdb \)
- \( L \rightarrow .55 \rightarrow bLb \)
- \( L \rightarrow .45 \rightarrow FQa \)
- \( F \rightarrow .4 \rightarrow bFb \)
- \( F \rightarrow .6 \rightarrow bab \)

\[ N_S = \{S, W, U, M, N, R, L, Q, F\} \]

\[ \Sigma_S = \{a, b, c, d\} \]
Table 2.1 \( G_S \), \( G_M \), \( G_A \), \( G_T \) for submedian, median, acrocentric, and telocentric, respectively.

\[
q_S(a|a) = .97 \quad q_S(b|a) = 0 \quad q_S(c|a) = 0 \quad q_S(d|a) = .008 \quad q_S(e|a) = 0.
\]
\[
q_S(b|b) = .963 \quad q_S(a|b) = 0 \quad q_S(c|b) = 0 \quad q_S(d|b) = .012 \quad q_S(e|b) = .003.
\]
\[
q_S(c|c) = .965 \quad q_S(a|c) = 0 \quad q_S(b|c) = 0 \quad q_S(d|c) = .003 \quad q_S(e|c) = .01.
\]
\[
q_S(d|d) = .955 \quad q_S(a|d) = .01 \quad q_S(b|d) = .01 \quad q_S(c|d) = .003 \quad q_S(e|d) = 0.
\]
\[
q_S(e|e) = .965 \quad q_S(a|e) = 0 \quad q_S(b|e) = .003 \quad q_S(c|e) = .01 \quad q_S(d|e) = 0.
\]
\[
q_D(a) = .01 \quad q_D(b) = .01 \quad q_D(c) = .01 \quad q_D(d) = .01 \quad q_D(e) = .01
\]
\[
q_{ij}(l|l) = .0024 \quad \text{for all } l, j, \quad l, j \in \{a, b, c, d, e\}
\]
\[
q_{ij}(l) = 0. \quad \text{for all } l \in \{a, b, c, d, e\}
\]

Table 2.2 Probabilities of terminal error transformations.
(B) Parsing-Time Speed-Up Technique

As discussed in [13], the time complexity of minimum-distance error-correcting parsing is $O(N^3)$. In a stochastic model, some of the probabilities associated with items in parse lists are so small that their corresponding derivations hardly ever occur. We may eliminate these unnecessary derivations by comparing the probability of an item with a lower bound before it is added to the parse list.

Let $[A \rightarrow \alpha \cdot \beta, i, p]$ be an item in $I_j$. We set $\xi$ as a lower bound. Then $[A \rightarrow \alpha \cdot \beta, i, p]$ is allowed to be added to $I_j$ only if $p > \xi$. A good selection of $\xi$ should depend on the tolerance of error density of a string. We use $C(A)^j - 1$ as the lower bound in this paper, where $A, C$ are constants. The value of $C$ should be less than the smallest deformation probability to permit its associated error to exist. The value of $A$ depending on stochastic grammar rules should be small enough to guarantee successful parsing of noisy strings.

For the string $y = cbabdbabcbabdbab$, Table 2.3 gives the approximate maximum number of errors allowed in a substring of given length when parsed by $G_M^1$ with $C = .001, A = .5$.

To illustrate the amount of parsing time saved by applying this technique, we parse a string both by $G_M^1$ with a preset lower bound and without a lower bound. Figure 2.9 shows this comparison in terms of the number of items in each parse list when string $cbabdbabcbabdbab$ is parsed by $G_M^1$ with the lower bound $A = .5, C = .001$. Total cpu time is 5.4 seconds with the preset lower bound and 34.3 sec. without.
Figure 2.9 Comparison of number of items in parse list of string cbabbdbbabcabbdbbab parsed by $G_1$ with and without preset lower bound
Table 2.3 Maximum number of errors allowed in substrings of cbabdbabcababdbab when parsed by $G_M'$ with lower bound .001 (.5)$^{j-1}$.

<table>
<thead>
<tr>
<th>length of substring</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum no. of error allowed</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

The trade-off of this speed-up is the generosity of MLECP. From Table 2.3, we notice that no string with consecutive errors can be successfully parsed in this example, but from $G_M'$, there is nearly 2.5% chance of consecutive errors for a string with string length 16. Therefore, the value of constant A and C should be carefully chosen such that the MLECP can meet a good compromise between its parsing time and error-correcting capability.

(C) Classification Result

Thirty-two test samples are generated from $G_S'$, $G_M'$, $G_A'$, and $G_T'$ with average string length 27, of which 20 are erroneous. The 32 samples are then tested by the Bayes classifier with lower bound set at .001 (.5)$^{j-1}$. The result is that, among 32 samples, 29 are correctly classified. The rest of the three samples are too noisy to be correctly classified due
Figure 2.10  cpu time vs. string length
to the use of lower bound of MLECP's. On the average, it takes 32.6 sec. to classify a string with an overall accuracy of 90.6%.

2.4.5. Sequential Classification of Noisy Patterns

In the previous section, we have designed MLECP's for the classification of noisy and distorted patterns. As it was shown, even if the lower bound of ECP are skillfully selected, the use of MLECP's is still too costly to be practically feasible. Persoon and Fu have proposed a sequential parsing scheme for strings generated from SCFG's [10]. By using an optimal decision rule and a suboptimal stopping rule, the parser scanned only part of the input string when a decision is made. For a given specified probability of error, \( \epsilon \), the sequential parsing scheme should be designed to minimize the average number of terminal symbols scanned.

The sequential classification algorithm (SCA) consists of a decision algorithm and a sequential parsing algorithm (SPA) which is essentially an extension of Earley's parser. Since errors are randomly distributed in a string, and SCA requires only part of the string to be parsed, chances are that the parsing of a noisy string by SCA would be successful before an error has ever been detected. Otherwise, a forced decision is made based on the information accumulated in the provision of SPA.

Let \( y = a_1a_2...a_j...a_n \) be a noisy input string. Suppose that we observed \( a_1a_2...a_j (j \leq n) \). The sequential parsing algorithm (SPA) computes \( p(a_1a_2...a_j|C_1) \) which is the probability that \( a_1a_2...a_j \) is a string in \( L(G_1) \), and \( p(a_1...a_jo|C_1) \) which is the total probability of strings in \( L(G_1) \) with \( a_1a_2...a_j \) as their prefix. By using these two quantities, a stopping rule tells when one has to stop observing more terminals, and
a decision rule assigns a class to \( y \) once the stopping rule indicates to stop.

The SCA designed for processing erroneous strings is similar to the algorithms presented in [10] for non-error-correcting parsing. A forced decision rule is added to the algorithms such that a decision can be reached where parsing is terminated by illegitimate terminals. The restriction on using \( \lambda \) productions in [10] is removed. The algorithms are given in Appendix B.

Two hundred test samples are generated from \( G'_S, G'_H, G'_A, \) and \( G'_T \). Among them, 112 are erroneous. The average string length is 26. The classification results of using SCA with a preset error bound \( \hat{e} \) are illustrated in Table 2.4 which shows classification accuracy and parsing time with respect to variable \( \hat{e} \). In the second experiment, the 200 test samples are passed through a non-sequential Earley's parser constructed from \( G_S, G_H, G_A, \) and \( G_T \). The result is given at the last row of Table 2.4. For the purpose of comparison, the third experiment uses 200 correct strings generated from SCFG's \( G'_S, G'_H, G'_A, \) and \( G'_T \) as test samples. They are classified by using SCA. The results are shown in Table 2.5. Curves showing accuracy vs. parsing time of Experiment 1, 2, and 3 are given in Figure 2.11.

From Table 2.5, if the parsed strings are error-free, the accuracy of sequential classification increases monotonically as the number of symbols scanned increases. One hundred percent accuracy may be reached when the error bound is sufficiently small. Whereas, the curve of accuracy vs. parsing time (or the number of parsed symbols per string) is convex when the parsed strings are noisy. In this case, the
Figure 2.11 Accuracy vs. parsing time of sequential classifier and nonsequential classifier
Table 2.4 Classification results from 200 strings (112 of them are erroneous)

<table>
<thead>
<tr>
<th>error bound $e$</th>
<th>no. of misclassification</th>
<th>accuracy</th>
<th>total cpu time (sec)</th>
<th>ave. no. of parsed symbol per string</th>
</tr>
</thead>
<tbody>
<tr>
<td>.0</td>
<td>33</td>
<td>83.5%</td>
<td>162.5</td>
<td>16</td>
</tr>
<tr>
<td>.03</td>
<td>26</td>
<td>87%</td>
<td>119.0</td>
<td>8</td>
</tr>
<tr>
<td>.05</td>
<td>26</td>
<td>87%</td>
<td>118.5</td>
<td>8</td>
</tr>
<tr>
<td>.1</td>
<td>27</td>
<td>86.5%</td>
<td>116.5</td>
<td>7</td>
</tr>
<tr>
<td>.15</td>
<td>27</td>
<td>86.5%</td>
<td>113.4</td>
<td>7</td>
</tr>
<tr>
<td>.18</td>
<td>30</td>
<td>85%</td>
<td>108.2</td>
<td>6</td>
</tr>
<tr>
<td>.21</td>
<td>30</td>
<td>85%</td>
<td>104.8</td>
<td>6</td>
</tr>
<tr>
<td>.25</td>
<td>32</td>
<td>84%</td>
<td>88.8</td>
<td>5</td>
</tr>
<tr>
<td>.3</td>
<td>35</td>
<td>82.5%</td>
<td>82.7</td>
<td>4</td>
</tr>
<tr>
<td>.35</td>
<td>57</td>
<td>71.5%</td>
<td>52.4</td>
<td>1</td>
</tr>
<tr>
<td>.4</td>
<td>57</td>
<td>71.5%</td>
<td>49.1</td>
<td>1</td>
</tr>
<tr>
<td>.5</td>
<td>93</td>
<td>53.5%</td>
<td>37.1</td>
<td>1</td>
</tr>
<tr>
<td>.6</td>
<td>101</td>
<td>49.5%</td>
<td>.036</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.5 Classification results from 200 correct strings.

Non-sequential Classifier

<table>
<thead>
<tr>
<th>error bound $e$</th>
<th>no. of misclassification</th>
<th>accuracy</th>
<th>total cpu time (sec)</th>
<th>ave. no. of parsed symbol per string</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>0</td>
<td>100%</td>
<td>128.2</td>
<td>8</td>
</tr>
<tr>
<td>.2</td>
<td>11</td>
<td>94.5%</td>
<td>110.2</td>
<td>7</td>
</tr>
<tr>
<td>.35</td>
<td>51</td>
<td>74.5%</td>
<td>50.0</td>
<td>1</td>
</tr>
</tbody>
</table>
accuracy of the sequential classifier cannot reach beyond a maximum point. In our example, the maximum is 87% accuracy when error bound is .05 with average parsing time .6 sec per string. The interpretation is intuitive, the more symbols are scanned, the higher the accuracy SCA can reach, and the better the chance an error is detected. As soon as an error is announced by the SPA of all the pattern grammars, a forced decision, whose low accuracy plays the counterpart, must be made.

To increase accuracy beyond the maximum accuracy of sequential classifier, we can use a sequential error-correcting parser as the classifier which is a SCA with its SPA constructed from error-induced grammars. The results of classifying 200 noisy strings are given in Table 2.6. When error bound is .15, the sequential error-correcting classifier reaches 94% accuracy with average 19.1 sec per string. Comparing with the result of 90.2% accuracy and average 32.6 sec per string using a non-sequential error-correcting classifier in Section 2.4.3, the sequential version achieved a slightly higher accuracy with less average parsing time.

The effectiveness of using SCA for processing noisy strings is largely pattern grammars dependent. It is noted that, most misclassifications of SCA occur between median and submedian because no distinctive difference appears at the first few symbols of their string representations. Whereas, it takes only two or three symbols to discriminate acrocentric from the other three types with very high accuracy. We may conclude that to increase both accuracy and efficiency using SCA, pattern grammars should be carefully constructed such that informative symbols will appear at the first few positions of derived strings.
<table>
<thead>
<tr>
<th>Error Bound</th>
<th>No. of Misclassification</th>
<th>Accuracy</th>
<th>Total CPU Time (sec)</th>
<th>Ave. No. of Parsed Symbol Per String</th>
</tr>
</thead>
<tbody>
<tr>
<td>.15</td>
<td>12</td>
<td>94%</td>
<td>3816</td>
<td>8</td>
</tr>
<tr>
<td>.25</td>
<td>27</td>
<td>86.5%</td>
<td>2650</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 2.6 Sequential error-correcting classification.
CHAPTER 3
ERROR-CORRECTING TREE AUTOMATA

3.1 Introduction

In applying syntactic methods to pattern recognition, one-dimensional (string) grammars are sometimes inefficient in describing two- or three-dimensional patterns. For the purpose of effectively describing high-dimensional patterns, high-dimensional grammars such as web grammars, graph grammars, and tree grammars have been proposed [50-52]. Properties of generalized finite automata, called tree automata, which accept finite trees of symbols as its input, have been studied by several authors [66-70]. Brainerd [66] proves that the class of systems which generates exactly the sets of trees accepted by the automata is a regular system. Fu and Bhargava [50] introduced the application of the tree systems into pattern recognition. In practical application, tree grammars and tree automata have been used in the classification of fingerprint patterns [38], the analysis of bubble chamber events [74], and the interpretation of LANDSAT data [39], etc.

The descriptive power of tree languages and the efficient analytical capability of tree automata made the tree system approach to pattern recognition very attractive. This chapter is concerned with the error-correcting version of tree automata. Unlike the string case, where the only relation between symbols is left-right concatenation, a tree structure would be deformed under deletion or insertion errors. The structure-
preserved error-correcting tree automaton (SPECTA) proposed in Section 3.3 takes only substitution errors into consideration. By introducing a blank element, a deletion error can be treated as substitution of a non-blank element by a blank element, and an insertion error becomes a non-blank element in substitution for a blank element. An example of using SPECTA in LANDSAT data interpretation is presented.

In Section 3.4, syntax errors on trees are defined in terms of five error transformations; namely, substitution, stretch, branch, split, and deletion. The distance between two trees is the least cost sequence of error transformations needed to transform one to the other. Based on this tree metric, a generalized error-correcting tree automaton (GECTA), is formulated, where transformations made on each terminal symbol are added to the system in the form of transition rules. An example of hand-printed character recognition is given to demonstrate the operation of GECTA.

3.2 Definitions

In this section, some basic definitions on trees, tree grammars, and tree automata given by Brainerd [66] are briefly reviewed.

**Definition 3.1.** Let \( N^* \) be the set of positive integers. Let \( U \) be the free monoid generated by \( N^* \). Let \( \cdot \) be the operation and 0 the identity of \( U \). The depth of \( a \in U \) is denoted \( d(a) \) and defined as follows: \( d(0)=0, d(a\cdot i)=d(a)+i, i \in N^+ \). \( a \leq b \) iff there exists \( x \in U \) such that \( a \cdot x=b \). \( a \) and \( b \) are incomparable iff \( a \nleq b \) and \( b \nleq a \).

**Definition 3.2.** \( D \) is a tree domain iff \( D \) is a finite subset of \( U \) satisfying (1) \( b \in D \) and \( a < b \) implies \( a \in D \), and (2) \( a \cdot j \in D \) and \( i < j \) in \( N^+ \) implies \( a \cdot i \in D \).
Definition 3.3. A rank is a pair \( \langle \Sigma, r \rangle \) where \( \Sigma \) is a finite set of symbols and \( r : \Sigma \rightarrow \mathbb{N} \). Let \( \Sigma_n = r^{-1}(n) \).

Definition 3.4. A tree over \( \Sigma \) (i.e., over \( \langle \Sigma, r \rangle \)) is a function \( \alpha : D \rightarrow \Sigma \), such that \( D \) is a tree domain and \( r[\alpha(a)] = \max\{1 | a \in D \} \).

i.e., the rank of a label at \( a \) must be equal to the number of branches in the tree domain at \( a \). The domain of a tree is denoted \( D(a) \) or \( D_a \).

Let \( T_\Sigma \) be the set of all trees over \( \Sigma \). The depth of \( \alpha \) is defined as,

\[
d(\alpha) = \max\{d(a) | a \in D_\alpha \}.
\]

Definition 3.5. Let \( a, b, b' \in U \) such that \( b = a \cdot b' \), then \( b/a = b' \).

\( b/a \) is not defined unless \( a \leq b \).

Definition 3.6. Let \( \alpha \in T_\Sigma \), and \( \alpha \in D_a \), \( a/\alpha = \{(b, x) | (a \cdot b, x) \in \alpha \} \), \( a/\alpha \) is the subtree of \( \alpha \) at \( a \) and \( a/\alpha \) occurs at \( a \) in \( \alpha \).

Definition 3.7. Let \( \alpha \in T_\Sigma \), \( \alpha \in U \), then \( a \cdot \alpha = \{(b, x) | (b/a, x) \in \alpha \} \).

Definition 3.8. Let \( \alpha \in D_a \), \( a, b \in T_\Sigma \), then \( \alpha(a + b) = \{(b, x) | (b/a, x) \in \alpha \} \cup a \cdot b \).

This is the result of replacing the subtree \( a/a \) at \( a \) by the tree \( b \).

Using postfix notation, the tree \( \alpha = \bigcup_{i=1}^{n} a_{i}a_{i+1} \cdot \cdots \cdot a_{n}x \).

Definition 3.9. \( t \) is a term over \( \langle \Sigma, r \rangle \) iff \( t = x \in \Sigma_0 \) or \( t = t_1 t_2 \cdots t_n x \) where \( x \in \Sigma_n \), and \( t_i, 1 \leq i \leq n \), is a term, \( T_\Sigma \) is the set of terms over \( \Sigma \). There is obviously a one-to-one correspondence between the terms over \( \Sigma \) and the trees over \( \Sigma \). If the corresponding tree of \( t \) is a subtree of \( \alpha \), we say \( t \) is a term in \( \alpha \).

Definition 3.10. A regular tree grammar over \( \langle \Sigma, r \rangle \) is a regular system \( G_t = (V, r', P, S) \) satisfying the following conditions:

(1) \( \langle V, r' \rangle \) is a finite-ranked alphabet with \( \Sigma \in V \), and \( r' | \Sigma = r \).

The elements in \( V \) and \( V - \Sigma \) are called terminal and nonterminal symbols, respectively.
(2) P is a finite set of productions of the form \( \phi \Rightarrow \psi \), where \( \phi \) and \( \psi \) are trees over \( <V, r'> \).

(3) \( S \subseteq T_V \) is a finite set of start symbols.

**Definition 3.11.** Derivation \( \alpha \Rightarrow \beta \) is in \( G_t \) iff \( \phi \Rightarrow \psi \) in \( P \) such that \( \alpha/\alpha = \phi \) and \( \beta = \alpha(\alpha+\psi) \). \( \alpha \Rightarrow \beta \) is a derivation iff there exist \( \alpha_0\alpha_1\ldots\alpha_m, m \geq 0 \) such that \( \alpha = \alpha_0\alpha_1\ldots\alpha_m = \beta \) in \( G_t \).

**Definition 3.12.** The language generated by \( G_t = (V, r', P, S) \) over \( <\Sigma, r> \) is defined as, \( L(G_t) = \{ \alpha \in T_\Sigma | \text{there exist } x \in S \text{ such that } x \Rightarrow \alpha \text{ in } G_t \} \).

**Definition 3.13.** A tree grammar \( G_t = (V, r', P, S) \) over \( <\Sigma, r> \) is expansive iff each production in \( P \) is of the form

\[
X_0 + \frac{x}{X_1 \ldots X_r(x)} \quad \text{or} \quad X_0 + x \quad \text{where } x \in \Sigma.
\]

and \( X_0, X_1 \ldots X_r(x) \) are nonterminal symbols.

Following the definition given by Brainerd [66], a tree automaton is a finite automaton with many-to-one state transition functions.

**Definition 3.14.** Let \( <\Sigma, r> \) be a rank and \( \Sigma = \{ \sigma_1, \sigma_2 \ldots \sigma_k \} \). A finite \( \Sigma \)-automaton or tree automaton over \( \Sigma \) is a system \( M_t = (Q, f_1 \ldots f_k, R) \) where

1. \( Q \) is a finite set of states,
2. for each \( i, 1 \leq i \leq k \), \( f_i \) is a relation on \( Q^{r(\sigma_i)} \times Q \),
3. \( R \subseteq Q \) is a set of final states.
if each \( f_i, 1 \leq i \leq k \), is a function \( f_i: Q^r(\sigma) \rightarrow Q \) then \( M_t \) is deterministic; otherwise \( M_t \) is nondeterministic.

The construction procedure of tree automaton for a regular tree grammar can be summarized as follows [50].

**Algorithm 3.1. Construction of tree automaton.**

**Step 1.** To obtain an expansive tree grammar \((V', r, P', S)\) for the given tree grammar \((V, r, P, S)\) over alphabet \( \Sigma \).

**Step 2.** The equivalent nondeterministic tree automaton is \( M_t = (V'-\Sigma, f_1 \ldots f_k, \{S\}) \) where \( f_i(x_1 \ldots x_n) - x_0 \) if \( x_0 \neq x_1 \ldots x_n \) is in \( P' \).

The acceptance of a tree by tree automaton is a backward procedure. It reads parallel branches simultaneously then transfers to the states of their immediate predecessors.

### 3.3 Structure-Preserved Error-Correcting Tree Automaton (SPECTA)

Let \( D \) be a tree domain, \( D \subset U \), \( \Sigma \) be a set of terminal symbols, we define \( T^D = \{ \alpha | \alpha \in T^D, D_\alpha = D \} \) be the set of trees in the tree domain \( D \).

In this section, substitution error is described in terms of the transformation \( S: T^D \rightarrow T^D \). For \( a \in D \), \( x \in \Sigma \), and \( \alpha, \alpha' \in T^D \), we write \( \alpha \leftarrow a/x \alpha' \) if \( \alpha' \) is the result of replacing the label on node \( a \) of tree \( \alpha \) by terminal symbol \( x \). Furthermore, \( S^k \) denotes the composition of \( S \) with itself \( k \) times.

The distance on trees in \( T^D \), \( d(\alpha, \alpha') \), is defined as the smallest integer \( k \) for which \( \alpha \leftarrow S^k \alpha' \) if \( \alpha \) and \( \alpha' \) are two trees in \( T^D \) for some \( D \subset U \). The function of \( d \) is symmetric and satisfies triangle inequality. Let \( L \) be a tree language, and tree \( \alpha' \notin L \). The essence of SPECTA is to search for a tree \( \alpha, \alpha \in L \) such that
\[ d(\alpha, \alpha') = \min_{\beta} \{d(\beta, \alpha') | \beta \in L, D_{\beta} = D_{\alpha'} \} \quad (3.1) \]

and reconstruct \( \alpha' \) as \( \alpha \). \( d(\alpha, \alpha') \) in the above equation is also defined as the distance of \( \alpha' \) from \( L \), denoting \( d(L, \alpha') \). \( \alpha \) is called the minimum distance correction of \( \alpha' \) in \( L \).

### 3.3.1 Minimum-Distance SPECTA

By adding terminal error production rules corresponding to substitution error transformations, the covering grammar \( G'_{t} = (V', r', P', S) \) of a given tree grammar \( G_{t} = (V, r, P, S) \) is constructed as follows:

**Step 1.** \( V' = (V - \Sigma) \cup \Sigma' \), where \( \Sigma' \supseteq \Sigma \) is a new set of terminal symbols.

**Step 2.** For each \( \gamma \in \Sigma' \) add to \( P' \)

\[
X_{0} \rightarrow \begin{cases} \gamma \vphantom{X_{1} \ldots X_{r}(x)} & \text{if } X_{0} \rightarrow X_{1} \ldots X_{r}(x) \in P \vphantom{X_{1} \ldots X_{r}(x)} \\ X_{1} \ldots X_{r}(x) \vphantom{X_{0} \gamma} & \text{if } X_{0} \rightarrow X_{1} \ldots X_{r}(x) \in P \vphantom{X_{1} \ldots X_{r}(x)} \end{cases}
\]

or \( X_{0} \rightarrow \gamma \) if \( X_{0} \rightarrow x \) is in \( P \).

The language generated from \( G'_{t} \) consists of the language \( L(G'_{t}) \) and its corresponding erroneous trees. Hence, \( L(G'_{t}) \) can be written as

\[
L(G'_{t}) = \{ \alpha' | \alpha' \in T_{\Sigma'}, \text{ and } \exists \alpha \in L(G_{t}) \text{ such that } D_{\alpha'} = D_{\alpha} \}
\]

For a given tree grammar, \( G_{t} \), the SPECTA is formulated to accept trees in \( L(G'_{t}) \) and to generate a parse that consists of the minimum number of error productions. Assume that \( \alpha' \) is an input tree, the operation of a SPECTA is a backward procedure of constructing a tree-like transition table from the frontiers to the root of \( \alpha' \). For each node \( a \in D_{\alpha'} \), there
is a corresponding set of triplets, denoting \( t_a \) in the transition table. Each triplet \((X, \ell, k)\) is added to \( t_a \) if \( X \) is a candidate state of node \( a \), \( \ell \) is the minimum number of errors in subtree \( a'/a \) when node \( a \) is represented by state \( X \), and \( k \) specifies the production rule used. The algorithm is given as follows.

**Algorithm 3.2. Minimum-distance SPECTA**

Input: \( G_t = (V, r, P, S) \) and tree \( a' \).

Output: Transition table of \( a' \) and \( d(L(G_t), a') \).

Method:

**Step 1.** If \( r[\alpha'(a)] = 0 \), \( \alpha'(a) = x \), then add to \( t_a \)

(a) \((X_0, 0, k)\) if \( X_0 + x \) is the \( k \)th rule in \( P \).
(b) \((X_0, 1, k)\) if \( X_0 + y \) is the \( k \)th rule in \( P \) and \( y \neq x \).

**Step 2.** If \( r[\alpha'(a)] = n > 0 \), \( \alpha'(a) = x \), then add to \( t_a \)

(a) \((X_0, \ell, k)\), if \( X_0 + x \) is the \( k \)th rule in \( P \) and

\[ (X_1, \ell_1, k_1) \in t_{a'1}, \ldots, (X_n, \ell_n, k_n) \in t_{a'n} \text{ then } \ell = \ell_1 + \ldots + \ell_n \]

(b) \((X_0, \ell, k)\), if \( X_0 + y \) is the \( k \)th rule in \( P \), \( y \neq x \), and

\[ (X_1, \ell_1, k_1) \in t_{a'1}, \ldots, (X_n, \ell_n, k_n) \in t_{a'n} \text{ then } \ell = \ell_1 + \ldots + \ell_n + 1 \]

**Step 3.** Whenever more than one item in \( t_a \) has the same state, delete the item with larger number of errors.

**Step 4.** If \((S, \ell, k) \in t_0\), then \( d(L(G_t), a') = \ell \). If no item is in \( t_0 \) of the form \((S, \ell, k)\), then no tree in \( L(G_t) \) is in tree domain \( D_a \), the input tree is rejected.

The minimum-distance correction of \( a' \) can easily be traced out from the transition table.
The tree grammar in the following example is a part of the highway grammar used in Section 3.3.3. In the meantime, we use it as an example here to illustrate the operation of SPECTA and to demonstrate the highway patterns recognition procedure that will be discussed in Section 3.3.3.

Example 3.1. Consider a set of vertical line patterns as given in Figure 3.1. Assume that elements in the 4 x 4 array are connected as a tree shown in Figure 3.2. Thus, each pattern has its corresponding tree representations. For example, pattern (b) in Figure 3.1 can be represented by the tree shown in Figure 3.3, where nodes labeled by symbol "b" represent blank or nonhighway elements "$\Box\$", and nodes labeled by "h" represent highway elements "$\mathbb{H}\$". The tree grammar that generates these tree representations can be written as:

$$G_H = (V, r, P, S)$$

over $\langle \Sigma, r \rangle$ where

$$V = \{S, A_0, A_1, A_2, A_3, X_0, l_1, l_2, l_3, \$, b, h\}$$

$$\Sigma = \{\$, b, h\}$$

$r(\$) = 1$, $r(b) = \{0, 1, 3\}$, $r(h) = \{0, 1, 3\}$

$$P: S \rightarrow \$$

(1), (2), (3), (4)

$$A_0$$

(5), (6), (7)

$$A_0^+$$

(8)

$$A_2^+$$

(9)
Given a noisy pattern shown in Figure 3.4(a), let $a'$ be its treerepresentation. $a'$ is given in Figure 3.4(b). The transition table of $a'$ resulted from using the minimum-distance SPECTA with respect to grammar $G_H$ is shown in Figure 3.5. Since $(S, 3, 2)$ is in $t_0$, $a'$ is accepted by the SPECTA and the number of errors in $a'$ is 3. Let the minimum-distance correction of $a'$ be called $\alpha$. The generation of $\alpha$ from the transition table is illustrated in Figure 3.6.

3.3.2 Maximum-likelihood SPECTA

When the probability distribution of patterns and the deformation probabilities on each terminal are available, error-correcting parsing based on maximum-likelihood criterion may provide a better recognition performance. Definitions of stochastic grammar, terminal deformation probabilities, and maximum-likelihood criterion have been introduced in Chapter 2. A similar approach to MLECP is used in formulating SPECTA based on maximum-likelihood criterion.
The expansive stochastic tree grammars and languages defined in [72] are briefly reviewed.

**Definition 3.15.** A stochastic tree grammar $G_s = (V,r,P,S)$ over $\Sigma$ is expansive if and only if each rule in $P$ is of the form

$$X_0 \rightarrow \frac{x}{X_1 \ldots X_r(x)}$$

or

$$X_0 \rightarrow x$$

where $x \in \Sigma$.

and $X_0, X_1 \ldots X_r(x) \in V-\Sigma$ are nonterminals.

**Definition 3.16.**

$$L(G_s) = \{(a,p(a)) | a \in T_\Sigma, S \Rightarrow a, i = 1 \ldots k, p(a) = \sum_{i=1}^{k} p_i\}$$

where $k$ is the number of all distinctly different derivation of $a$ from $S$, and $p_i$ is the probability associated with the $i$th distinct derivation of $a$ from $S$.

Assume that the occurrence of substitution error on a terminal is independent from its neighboring terminals. Fung and Fu [31] define a substitution error made on strings to be a stochastic mapping $\sigma: \Sigma \rightarrow \Sigma$ such that $\sigma(a) = b$, if $a$ and $b \in \Sigma$, with probability $q(b|a)$ and furthermore,

$$\sum_{b \in \Sigma} q(b|a) = 1$$

(3.2)

The same definition can be applied to model a substitution error made on trees. Furthermore, assume that $t = t_1 \ldots t_n x$ is a term over $<\Sigma,r>$,

$$\sigma(t_1 \ldots t_n x) = \sigma(t_1) \ldots \sigma(t_n)\sigma(x)$$

(3.3)

If two trees $a = t_1 \ldots t_n x$, $a' = t'_1 \ldots t'_n x'$ are both in $T_\Sigma^D$, the probability of $a'$ being the noisy deformed tree of $a$ is
Figure 3.1 Vertical Line Patterns

Figure 3.2 A Tree Structure

Figure 3.3 Tree Representation of Pattern (b)
Figure 3.4 A Noisy Pattern (a) and Its Tree Representation, a', (b).
Figure 3.5 Transition Table of Tree α¹
Figure 3.6  The generation of $a$, the correction of $a'$
We further have
\[ \sum_{\alpha' \in T_{\Sigma}^D} q(\alpha'|\alpha) = 1 \quad \text{for any } \alpha \in T_{\Sigma}^D, \text{D} \subseteq \text{U} \quad (3.5) \]

For a given stochastic tree grammar \( G_s = (V, r, P, S) \) over \( \langle \Sigma, r \rangle \), when the deformation probabilities, \( q(y|x) \), are known for all \( x \in \Sigma \), and \( y \in \Sigma' \), the stochastic expanded grammar is \( G'_s = (V', r', P', S) \) over \( \langle \Sigma', r' \rangle \) where \( V' = (V-\Sigma) \Sigma' \) and \( \Sigma' \supseteq \Sigma \) is the set of terminal symbols, and for all \( y \in \Sigma', X_0 \overset{P'}{\rightarrow} y \text{ is in } P' \) if \( X_0 \overset{P}{\rightarrow} x \text{ is in } P \), or \( X_0 \overset{P}{\rightarrow} y \)
\[ x \]
\[ X_1 \cdots X_r(x) \]

is in \( P' \), if \( X_0 \overset{P}{\rightarrow} x \text{ is in } P \) and \( p' = p q(y|x) \).

The language that generated from \( G'_s \) can be written as:
\[ L(G'_s) = \{(a', p'(a')) | a' \in T_{\Sigma}', p'(a') = \sum_{\alpha \in L(G_s)} q(\alpha'|a)p(a) \} \quad (3.6) \]

Suppose that the given noisy input tree \( a' \) is in tree domain \( D \), the maximum-likelihood decision rule in this case is to choose a tree \( \alpha \) in \( L(G_s) \) of domain \( D \), i.e., \( \alpha \in L(G_s) \) and \( \alpha \in T_{\Sigma}^D \), such that
\[ q(\alpha|a)p(a) = \max_\beta \{q(\alpha'|\beta)p(\beta) | \beta \in L(G_s) \cap T_{\Sigma}^D \} \quad (3.7) \]

We call this value, \( q(\alpha'|a)p(a) \), the probability of \( a' \) being a noise deformed tree of \( L(G_s) \) and denote it as \( q(a'|G'_s) \).

The structure-preserved maximum-likelihood SPECTA is given as follows:
Algorithm 3.3. Maximum-likelihood SPECTA

Input: (1) Stochastic tree grammar $G_{s} = (V, r, P, S)$ over $<\Sigma, r>$.

(2) Defoliation probabilities, $q(y|x)$, for all $x \in \Sigma$, $y \in \Sigma'$.

(3) Input tree $a'$.

Output: Transition table of $a'$ and $q(a'|G_{s}')$.

Method:

**Step 1.** If $r[a(a)] = 0$, $a(a) = y$ then add to $t_{a}$, $(X_{0}, p', k)$, if $X_{0} \xrightarrow{p} x$ is the $k$th rule in $P$ and $p' = p \cdot q(y|x)$.

**Step 2.** If $r[a(a)] = n$, $n > 0$, $a(a) = y$ then add to $t_{a}$, $(X_{0}, p', k)$, if $X_{0} \xrightarrow{p} x$ is the $k$th rule in $P$ and $(X_{1}, p'_1, k_1)$, $\ldots$, $(X_{n}, p'_n, k_n)$, then $p' = p'_1 \cdots p'_n \cdot q(y|x)$.

**Step 3.** Whenever more than one item in $t_{a}$ have the same state, delete the item associated with smaller probability.

**Step 4.** If $(S, p', k) \in t_{0}$, then $q(a|G_{s}') = p'$. If no item in $t_{0}$ is associated with the start state $S$, then no tree in $L(G_{s})$ is in tree domain $D_{a}$. Input tree is rejected.

3.3.3 Application of SPECTA to LANDSAT Data Interpretation

Recently, syntactic methods have been used to analyze and interpret data obtained from the earth resource technology satellite (LANDSAT) [39,53]. The input data used in [39,53] are the results of pointwise classification [73]. Each pixel collected by LANDSAT represents a ground area of approximately $60 \times 70$ m$^2$. According to spectral and/or temporal measurements of the object, a pixel is then classified into classes of water, cloud, downtown, concrete, or grass, etc. Due to the resolution size, spectral signals of smaller objects are usually composed of
Figure 3.7 Pointwise classified highway data obtained from Grand Rapids, Michigan.
Figure 3.8 Divided highway map of northern part of Grand Rapids.
reflectance of several different kinds of ground cover. For instance, the spectral signal of a segment of a highway actually results from a combined reflectance of concrete surface, grass, and transportation vehicles. Consequently, the variation of size of smaller objects and their surroundings changes their reflectances, and thus, their spectral properties from point to point. This uncertainty causes some difficulty in setting threshold for classification based on spectral information of individual points only. One example of the results of pointwise classified highway patterns is given in Figure 3.7 which covers the area of the northern part of Grand Rapids, Michigan. Each symbol "H" represents a pixel that is classified as a segment of highway. Figure 3.8, which is obtained from the official highway map, indicates the major divided highways of the same area.

As illustrated in Figure 3.7, the inadequate resolution of highways and the mass of scattered concrete and grass-mixed objects other than highways result in discontinuity of highways and spurious points from pointwise classification. Syntactic methods have the advantage of using contextual and structural information contained in patterns for recognition purpose. In order to discriminate highways or rivers from other objects having similar spectral properties a tree system approach is proposed by Li and Fu [39]. The method demonstrates fairly good results in recognizing rivers among watery areas and some modern buildings with glass walls having similar reflecting surfaces as water, but poor in analyzing highway patterns. Evidently, highway patterns in some cases are too noisy to be effectively analyzed by a conventional training and parsing methods.

In this experiment, an input picture is processed window by window, where window size is 8 x 8 array of pixels. The labels on single pixels
are used as primitives. Thus, we have two kinds of primitives: "h" represents a highway pixel and "b" represents a blank or nonhighway pixel. Initially, sample patterns of vertical, horizontal, or diagonal line are used as our positive samples (or desired patterns). Assume that only a single segment of highway or at most two intersected highways that appeared within a window is considered. Hence, some patterns consisting of two such sample patterns are also added to the set of positive samples.

Similar to the procedure illustrated in Example 3.1, an array of primitives in a window are represented as a tree. Each primitive becomes a labeled node in the tree representation. We fix the tree domain to be \( D_H \), and allow node label to be either "h" or "b". Totally there are \( 2^64 \) tree representations in \( T_\Sigma^D_H \), where \( \Sigma = \{b,h,\$\} \), and \$\) is the start terminal. Hence, the set of all possible patterns in an 8 x 8 window and the set of all labeled trees in \( T_\Sigma^D_H \) are one-to-one correspondence. Figure 3.9 illustrates the correspondence between points in the 8 x 8 array and nodes in the tree domain \( D_H \). The set of positive sample patterns can now be transformed into a set of positive sample trees of domain \( D_H \). The construction of a pattern grammar, when error-correcting parser is used as recognizer, is a little different from the regular procedure [73]. With some presumed patterns as positive samples, the language generated from the constructed grammar must be as close to the set of positive samples as possible. Otherwise, there is always the possibility that the noisy pattern finds its best match in the set of "unwanted sentences" (sentences in \( L(G) \) but not in the set of positive samples).

The tree grammar, \( G_H \), constructed to generate positive sample trees is given in Appendix C. Figure C.1 in Appendix C illustrates some
Figure 3.9 Nodes in tree domain $D_H$ and their corresponding positions in the $8 \times 8$ array.
typical patterns whose tree representations are in $L(G_H)$. In Figure C.1, "Group A" denotes the group of patterns that are generated from rules of the form

$$S \rightarrow \mathcal{A}_1, \quad I = 1 \ldots 8, \quad \text{"Group B" are from } S \rightarrow \mathcal{B}_1 \text{ etc.}$$

Note that the tree whose nodes are all labeled with "b" also belongs to $L(G_H)$. Actually, the pattern it represented is an undesired one, i.e., a window with no highway passing through. It is our negative sample tree, denoting as $\lambda$. We may categorize the sets of trees that have been introduced so far as follows: $T^D_H$ is the universe we are working on, where $E = \{\$, h, b\}$ and $\$ is the start terminal. $T^D_H \cap (L(G_H) - \{\lambda\})$ is the positive sample set, denoting as $S^+$. $\{\lambda\}$ is the negative sample set denoting as $S^-$ [74]. Let the set $T^D_H \cap (S^+ \cup S^-)$ be denoted as $N$, then $N = T^D_H \cap L(G_H)^c$. Apparently, elements in $N$ are noisy patterns, which are normally unrecognizable using a conventional parsing scheme. The idea of using error-correcting tree automaton is to measure the distance between an input pattern and patterns in $S^+ \cup S^-$. If the input pattern is in $N$, it will further be reconstructed to its best matching pattern in terms of the minimum number of mislabeled nodes, in $S^+$, or erased if its best matching pattern is in $S^-$. Since $S^-$ is also in $L(G_H)$, we may measure the distance of the input tree with $S^+$ and $S^-$ at the same time.

Assume that an input picture is scanned column by column from left to right. After an $8 \times 8$ array of primitives has been processed by SPECTA, we erase the top left $5 \times 5$ array of pixels, i.e., replace "H" points by
blanks of the original pattern and superimpose the reconstructed pattern on the original window. The scanning window then moves down five rows of pixels so that the pattern appeared in the window is processed by the same procedure. After eight columns of pixels have been scanned, the process is repeated from the sixth column. By doing this, the error-correcting scheme would be furnished with contextual information, not only within the window, but between the window and its surroundings. Using the pattern shown in column 141-148, row 1-18 of Figure 3.7, the window-by-window correcting process is illustrated in Figure 3.10. The flowchart of the entire recognition procedure is given in Figure 3.11.

This error-correcting scheme of the highway recognition problem is programmed in Fortran IV on CDC 6500 computer and tested using the data shown in Figure 3.7. The result is shown in Figure 3.12. There are 80 x 160 pixels in the input data. The cpu time for processing is 150 sec.

We also use the grammar $G_H$, which is trained from Grand Rapids data to analyze some other noisy data, such as data obtained from Lafayette, Indiana. The pointwise classified data of Lafayette is shown in Figure 3.13 which contains 125 x 125 pixels. The result of the error-correcting analysis is shown in Figure 3.14. The cpu time used is 101 sec. For comparison, we use the highway map shown in Figure 3.15 as ground truth.

There are several remarks to be made about this highway recognition example. (1) Originally, the presumed positive samples were more than those generated from $G_H$ in Appendix C. Many patterns of two intersected highways are considered. After the originally constructed grammar was tested by Grand Rapids data, we further removed production rules that were infrequently (or not) accessed and reduce the highway grammar to $G_H$ in Appendix C.
Figure 3.10 Window-by-window correcting process
Read input data into an array
INPUT (n, m)

1 = 1, j = 1

Pixels in the 8 x 8 window is connected as a tree of domain D_h, namely 

Parse the tree by ECTA

if 
   \[ d(L, R) > k \]

no 

yes 

Re-label the tree to be tree \( \lambda \)

Trace back the parse and re-label the tree

INPUT (i, k) = blank
for \( i = 1 \ldots l + 4 \)
\[ k = j \ldots j + 4 \]

Superimpose the re-labeled tree on INPUT (i, k)
for \( i = 1 \ldots l + 7 \)
\[ k = j \ldots j + 7 \]

\[ i = i + 5 \] 

\[ j = j + 5 \]

if 
\[ i = n - 4 \]

yes 

no 

if 
\[ j = m - 4 \]

yes 

STOP

Figure 3.11 Flow chart of highway recognition using SPECTA
Figure 3.12 SPECTA processed result of the Grand Rapids area.
Figure 3.13 Pointwisely classified data of the Lafayette area.
Figure 3.14 SPECTA processed result of the Lafayette area.
Figure 3.15 Highway map of Lafayette, Indiana.
(2) An alternative solution, if not using the error-correcting scheme, is to obtain a large set of positive samples and negative samples from training data, and to construct a grammar so that all the negative samples are excluded from the language and all the positive samples are included. It is difficult to construct such a grammar when patterns are irregular. Besides, due to the large variation of input data, a slight difference in the training set will cause some patterns to be rejected during parsing.

(3) For our convenience, we use fixed tree domain. A deleted segment of highway is taken as a substitution of a highway primitive by a blank primitive. A spurious H point is considered as substitution of a blank primitive by a highway primitive. A generalized ECTA which handles substitution errors as well as deletion and insertion errors is introduced in the following section.

3.4 Generalized Error-Correcting Tree Automaton

3.4.1 Distance on Trees

In Section 3.3, since only substitution transformations are considered, tree distance is measurable when two trees are in the same domain. Our purpose in this section is to define transformations between trees in different tree domains such that tree distance is measurable for any two given trees. Consequently, the transformations we defined can be applied to a tree automaton.

Errors on trees are defined to be of the following five types:

(1) the substitution of the label of a node by another terminal symbol,

(2) the insertion of an extraneous labeled node between a node and its immediate predecessor.
(3) the insertion of an extraneous labeled node to the left of all the immediate successors of a node,

(4) the insertion of an extraneous labeled node to the right of a node,

(5) the deletion of a node of rank 1 or 0.

The three operations of insertions in rule (2), (3) and (4) are named as stretch, branch, and split, respectively, according to the relative position of the inserted node to the original tree. Apparently, the inverse operation of any type of insertion is deletion, and the inverse of deletion operation is one of the three types of insertion.

We define five transformations $S, T, P, B, D$, from $T_{\Sigma}$ to the subsets of $T_{\Sigma}$, to describe substitution, stretch, split, branch, and deletion errors respectively. Let $a$ be a tree over $<\Sigma, r>$, $a \in \mathbb{U}$, $y \in \Sigma$, and $c \cdot 1 = a$, $r[a(c)] = n$, then error transformations are defined as follows:

1. $S_{a/y}(a) = a(a+(0,y))U(1 \cdot a/a \cdot i \mid 1 < i < r[a(a)])$.
2. $T_{a/y}(a) = a(a+(0,y))U(1 \cdot a/a)$. 
3. $B_{a/y}(a) = a(a+(0,a(a)), (1,y))U(1 \cdot 1 \cdot a/a \cdot i \mid 1 < i < r[a(a)])$.
4. $P_{a/y}(a) = a(c \cdot n+1 + a/c \cdot n) \ldots (c \cdot n+1 + a/c \cdot n)U(a+(0,y))$.
5. $D_{a/y}(a) = a(c \cdot 1 + a/c \cdot 1+1) \ldots (c \cdot 1 + a/c \cdot n)U(c \cdot n+\lambda)$ if $r[a(a)] = 0$
   \[
   \begin{cases} 
   a(c + a/a) & \text{if } r[a(a)] = 1 
   \end{cases}
   \]

$S, T, B, P$ and $D$ transformations are illustrated in Figure 3.16 (a), (b), (c), (d) and (e), respectively.

We write $a \xrightarrow{\Delta} \beta$, if $\beta$ is in $\Delta(a)$, where $\Delta \in \{S, T, B, P, D\}$, and further denote that $a \xrightarrow{\Delta^k} \beta$ for $k \geq 0$, if $\beta$ is derived from $a$ by applying
Figure 3.16 S, T, B, P, and D transformations on $\alpha$. 
k transformations, where $\Delta^k$ denotes the composition of $\Delta$ with itself $k$ times. The distance on trees over $\Sigma$, $d(\alpha, \beta)$, is defined as the smallest integer $k$ for which $\alpha \xrightarrow{\Delta^k} \beta$, if $\alpha$ and $\beta$ are two trees in $T_\Sigma$.

For example, given two trees $\alpha$ and $\beta$, where $\alpha = \{(0,\$), (0,-), (1$1,P), (2,-), (2$1,q)\}$. $\beta = \{(0,\$), (1,V), (1$1,P), (1$2,q), (2,q)\}$, then $d(\alpha, \beta) = 3$, since $\beta = D_2/ (P_1$1/q (S_1V(\alpha))) and no other derivation of $\beta$ from $\alpha$ costs transformations less than three. Trees $\alpha$ and $\beta$ are shown in Figure 3.17.

Let $L$ be a tree language, a tree $\beta$ not in $L$ can be derived from some tree in $L$ by a sequence of error transformations. The distance between $\beta$ and $L$ is defined as,

$$d(L, \beta) = \min_{\alpha} \{d(\alpha, \beta) \mid \alpha \in L\}$$

(3.8)

Example 3.2. Assume a directed graph of labeled vertices and unlabeled branches as shown in Figure 3.18, the tree grammar constructed to generate such patterns is given as follows:

$$G_t = (V, r, P, S)$$

$V = \{S,A,B,C,D,E,H,\$,a,b,c,d,e,h\}$

$r(\$$) = \{1\}, r(a) = \{0,3\}, r(b) = \{0,3,4\}, r(c) = \{0,2\}$

$r(d) = \{0,2,3\}, r(e) = \{1,2\}, r(h) = \{1,2\}$

$$S \xrightarrow{\$}$$

$$A \xrightarrow{a}$$

$$H \xrightarrow{D} B$$

$$H \xrightarrow{a}$$

$$\alpha$$
Suppose that $a'$ is the given distorted pattern. The successive graphs after each tree error transformation is applied are illustrated in Figure 3.19.

### 3.4.2 The Formulation of GECTA

For any given $a'$ not in language $L$, the generalized error-correcting tree automaton (GECTA) is formulated to search for $a$ in $L$ such that the distance of $a$ from $a'$ is the smallest among all the sentences in $L$. That is,

$$d(a,a') = \min_{\beta \in L} d(\beta, a')$$

Note that the condition $D_\beta = D_{a'}$ in equation (3.1) is removed here.

Before introducing the algorithms for GECTA, a normal form of trees and tree grammars, called the binary form, is defined.
Figure 3.17 $\beta = D_2/\langle p_{1/2} (S_1 \circ (a)) \rangle$.

Figure 3.18 A directed graph
Figure 3.19 The sequence of transformation of $a'$ from $a$.
Figure 3.19 Continued
\[ \text{Figure 3.19 Continued} \]

\[ d(L, \alpha') = d(\alpha, \alpha') = 4 \]
(A) Tree Binary Form

A binary tree grammar is defined as follows:

**Definition 3.17.** A tree grammar \( G_b = (V_b, r_b, P_b, S) \) over \( \Sigma_b, r_b \) is said to be in binary form if,

1. A pseudo symbol \( * \) is in \( \Sigma_b \).
2. \( r_b = \{0,1,2\} \)
3. Each production in \( P_b \) is in one of the following forms:
   - (a) \( U_1 + X_1 * \).
   - (b) \( U_2 + U_1 X_1 * \)
   - (c) \( X_0 + U_1 X_1 x \)
   - (d) \( X_0 + X_1 x \)
   - (e) \( X_0 + x \)

where \( U_1, U_2, X_0, X_1 \) are in \( V_b - \Sigma_b, x \in \Sigma_b - \{\ast\} \).

Let \( \alpha \) be a tree in \( T_\Sigma \), and \( * \) be a pseudo symbol not in \( \Sigma \), the conversion of tree \( \alpha \) into its binary form \( \alpha^* \) is given in the following Algorithm.

**Algorithm 3.4.** Binary Form Conversion.

Input: Tree \( \alpha \) in \( T_\Sigma \).
Output: \( \alpha^* \), binary form of \( \alpha \).

Method: Repeat CONVERT until \( \alpha \) is in binary form.

<CONVERT>: (1) If \( t_0 = x \) is a term in \( \alpha \), then \( t_0 \) is said to be in binary form.

(2) If \( t_1 \ldots t_n \) are already in binary form, and \( t_0 = (t_1 \ldots t_n) x \) is a term in \( \alpha \), \( x \in \Sigma_n \) then \( t_0 \) is said to be in binary form if \( n = 1 \), then \( t_0 = (t_1) x \).
If \( n > 1 \), then define \( t_1^*...t_{n-1}^* \) such that

\[
\begin{align*}
t_1^* &= (t_1)^* \\
t_2^* &= (t_1^* t_2)^* \\
&\vdots \\
t_{n-1}^* &= (t_{n-2}^* t_{n-1})^* \\
t_0^* &= (t_{n-1}^* t_n)^* 
\end{align*}
\]

The construction of the binary tree grammar \( G_b = (V_b, P_b, r_b, S) \) from a given tree grammar \( G_t = (V, r, P, S) \) in expansive form is as follows:

Let \( E_b = E \cup \{ \ast \} \), add \( V \) to \( V_b \). \( P_b \) is constructed as follows:

**Step 1**. For each production \( \ast \in P \) of the form \( X_0 \ast X, x \in E_0 \), add \( X_0 \ast x \) to \( P_b \) and \( r_b(x) = 0 \).

**Step 2**. For each production \( \ast \in P \) of the form \( X_0 \ast X_1 x, x \in E_1 \), add \( X_0 \ast X_1 x \) to \( P_b \) and \( r_b(x) = 1 \).

**Step 3**. For each production \( \ast \in P \) of the form \( X_0 \ast X_1 \ldots X_n x, x \in E_n, n \geq 2 \), add,

\[
\begin{align*}
U_{01} &+ X_1^* \\
U_{02} &+ U_{01} X_2^* \\
&\vdots \\
U_{0n-1} &+ U_{0n-2} X_{n-1}^* \\
X_0 &+ U_{0n-1} X_n x \\
\end{align*}
\]

to \( P_b \), and \( r_b(x) = \{1, 2\} \), \( r_b(x) = 2 \). Add \( \{U_{0i} \mid 1 \leq 1 \leq n-1\} \) to \( V_b \).
Let the binary form of a tree language \( L(G_t) \) be denoted as \( L_b(G_t) \) and \( G_b \) be the binary form of grammar \( G_t \), then \( L_b(G_t) = L(G_b) \).

**Example 3.3. Character E**

A character E is shown in Figure 3.21 (a). Based on the pattern primitives defined in Figure 3.20, the tree representation \( a \) of E is shown in Figure 3.21 (b). The tree grammar \( G_t \) which generates the character E of different sizes is as follows:

\[
G_t = (V, r, P, S) \text{ over } \langle \Sigma, r \rangle
\]

\[
V = \{S, B, C, D, $, b, d\}, \Sigma = \{\$, b, d\}
\]

\[
r($) = 2, \ r(b) = \{1,2\}, \ r(d) = \{0,1\}
\]

\[
P:
\]

\[
S \rightarrow S \backslash B \ D
\]

\[
B \rightarrow b \backslash C \ D, \ b
\]

\[
D \rightarrow d \backslash D, \ d
\]

\[
C \rightarrow b \backslash C, \ b \quad D
\]

The binary form of \( a, a^* \), is given in Figure 3.22 and the binary form of \( G_t, G_b \) is:

\[
G_b = (V_b, r_b, P_b, S) \text{ over } \langle \Sigma_b, r_b \rangle
\]

\[
V_b = \{S, U_S, B, U_B, D, C, $, b, d, *\}, \quad \Sigma_b = \Sigma \cup \{*\}
\]

\[
r_b($) = \{2\}, \ r_b(b) = \{1,2\}, \ r_b(d) = \{0,1\}, \ r_b(*) = \{1\}.
\]
Figure 3.20 Pattern primitives

Figure 3.21 Character E (a) and its tree representation (b)
Figure 3.22 $a^*$, binary form of $a$
Let \( M_b = (V_b, \Sigma_b, F, \{S\}) \) be a tree automaton that accepts a tree grammar in binary form \( G_b = (V_b, \Sigma_b, F, \{S\}) \) over \( \Sigma_b \), where \( F \) is the set of transition functions on \( (V_b, \Sigma_b, r_b, P_b, S) \) for all \( x \in \Sigma_b \). By adding error transitions according to the transformations defined in Section 3.4.1, the expanded tree automaton that accept all the possible erroneous trees is constructed as follows:

**Algorithm 3.5. Expanded Tree Automaton \( \dot{M}_b \).**

**Step 1.** \( \dot{M}_b = (\bar{V}, \bar{F}, \{S\}) \) where \( \bar{V} = (V_b, \Sigma_b, \{1\}) \), \( \bar{F} = F \cup F^s \cup F^d \cup \{F_1\} \), and \( F^s, F^d, \) and \( F_1 \) are called substitution, deletion, and insertion error transitions, respectively.
Step 2. \( F^S = \{ f_y(X_0) \sim X_0, \sigma | f_x(\xi) \sim X_0 \in \mathcal{F}, x \neq \star, \gamma \in \Sigma_b - \{\star\}\) \)

Step 3. \( F^D = \{ f(\xi) \sim X_0, \gamma | f_x(\xi) \sim X_0 \in \mathcal{F}, x \neq \star\} \cup \{ f(\xi) \sim X_0, 0 | f_x(\xi) \sim X_0 \in \mathcal{F}\} \)

Step 4. 

(a) Add \( f_y(X_0) \sim X_0, \delta, \) to \( F^1 \) for all \( y \in \Sigma_b - \{\star\} \) if \( f_x(\xi) \sim X_0 \) is in \( \mathcal{F} \) and \( x \neq \star \).

(b) Add \( f_x(l, X_1) \sim X_0, 0, \) and \( f_y(l, X_1) \sim X_0, \sigma \) or \( f_x(l) \sim X_0, 0, \) and \( f_y(l) \sim X_0, \sigma \) to \( F^1 \) if \( f_x(X_1) \sim X_0 \) or \( f_x \sim X_0 \) is in \( \mathcal{F}, X \in \Sigma_b \).

(c) Add \( f_y(X_0) \sim X_0, 0, f_x(X_0, l) \sim X_0, 0, f_y(X_0, l) \sim X_0, \sigma \) and \( f_x(X_0, l) \sim X_0, 0 \) to \( F^1 \) if \( f_x(\xi) \sim X_0 \) is in \( \mathcal{F} \) for all \( x \in \Sigma_b \).

(d) Add to \( F^1, f_y(l) \sim l, \delta, \) and \( f_y(l, l) \sim l, \delta, \) for all \( y \in \Sigma_b \) and \( f_y \sim l, \delta, \) for all \( y \in \Sigma_b - \{\star\} \)

Where \( \xi \in (V_b - \Sigma_b)^I \), \( i = 1, 2 \).

The notations \( \sigma, \gamma, \) and \( \delta \) associated with transitions represent costs of substitution, deletion, and insertion errors, respectively. Hence, weighted distance can be measured by using expanded tree automaton.

(a), (b), and (c) in Step 4 introduce stretch, branch, and split operations, respectively.

The search algorithm for the least cost (minimum-distance) solution is to construct a tree-like transition table with all candidate states and their corresponding costs recorded. Each element in the transition table corresponds to a node in the tree domain of the input tree. Let it be denoted
as $t_a$ if $a \in D_a$, and $a'$ be the input tree, then a pair $(X,n)$ is in $t_a$ if $X$ is a candidate state representing subtree $a'/a$, $n$ is the minimum cost associated with $X$, the algorithm is given as follows:

**Algorithm 3.6. Minimum-Distance GECTA**

Input: Expanded Tree Automaton $\tilde{M}_b = (\tilde{V}, \tilde{F}, \{S\})$ of a tree language $L(G_t)$, and input tree $a'$.

Output: $d(L(G_t), a')$, and transition table of $a'$.

Method:

**<SUBTREE(Start)>**

(a) $\text{SUBTREE}(\text{Start}) = \{(X_0, \gamma) \mid f \sim X_0, \gamma \in F_D\}$.

(b) Add $(X_0, n)$ to $\text{SUBTREE}(\text{Start})$ if $f(X_1) \sim X_0$, $\gamma \in F_D$, $(X_1, \pi) \in \text{SUBTREE}(\text{Start})$, then $n = \gamma + \pi$.

(c) Add $(X_0, n)$ to $\text{SUBTREE}(\text{Start})$ if $f(X_1, X_2) \sim X_0$, $\gamma \in F_D$, $(X_1, \pi_1)$ and $(X_2, \pi_2) \in \text{SUBTREE}(\text{Start})$, then $n = \gamma + \pi_1 + \pi_2$.

(d) Whenever two or more items associated with the same state. Delete items with higher costs.

**<SUBTREE(X, e)>**

(a) $\text{SUBTREE}(X, e) = \{(X_0, n) \mid f(X) \sim X_0, \gamma \in F_D, n = \gamma + e\} 
\cup \{(X, e)\}$.

(b) Add $(X_0, n)$ to $\text{SUBTREE}(X, e)$ if $f(Y) \sim X_0, \gamma \in F_D$, $(Y, \pi) \in \text{SUBTREE}(X, e)$, then $n = \gamma + \pi$.

(c) Add $(X_0, n)$ to $\text{SUBTREE}(X, e)$ if $f(Y_1, Y_2) \sim X_0, \gamma \in F_D$, $(Y_1, \pi_1) \in \text{SUBTREE}(\text{Start})$ and $(Y_2, \pi_2) \in \text{SUBTREE}(X, e)$, or $(Y_1, \pi_1) \in \text{SUBTREE}(X, e)$ and $(Y_2, \pi_2) \in \text{SUBTREE}(\text{Start})$, then $n = \gamma + \pi_1 + \pi_2$. 
(d) Delete redundant states.

**Step 1.** If \( r[\alpha(a)] = 0, \alpha(a) = x \), then

(a) \( (X_0, n) \) is in \( t_a \) if \( f_x(X) \sim X_0, \lambda \in F \cup F^S \cup F^I \), for all \((X, \pi)\) in SUBTREE(Start), \( n = \lambda + \pi \).

(b) \( (X_0, n) \) is added to \( t_a \) if \( f_x(X_1, X_2) \sim X_0, \lambda \in F \cup F^S \cup F^I \), \((X_1, \pi_1) \) and \((X_2, \pi_2) \) \( \in \) SUBTREE(Start), \( n = \lambda + \pi_1 + \pi_2 \).

(c) Delete redundant states.

**Step 2.** If \( r[\alpha(a)] = 1, \alpha(a) = x \), then

(a) \( (X_0, n) \) is in \( t_a \) if \( f_x(Y) \sim X_0, \lambda \in F \cup F^S \cup F^I \), for all \((Y, \pi)\) \( \in \) SUBTREE \((X, \theta)\) and \((X, \theta) \) \( \in \) \( t_a \), \( n = \lambda + \pi \).

(b) \( (X_0, n) \) is added to \( t_a \) if \( f_x(Y_1, Y_2) \sim X_0, \lambda \in F \cup F^S \cup F^I \), for all \((Y_1, \pi_1) \) \( \in \) SUBTREE \((X, \theta)\) and \((Y_2, \pi_2) \) \( \in \) SUBTREE \((X, \theta)\) where \((X, \theta) \) \( \in \) \( t_a \), then \( n = \lambda + \pi_1 + \pi_2 \).

(c) Delete redundant states.

**Step 3.** If \( r[\alpha(a)] = 2, \alpha(a) = x \), then

(a) \( (X_0, n) \) is in \( t_a \) if \( f_x(Y_1, Y_2) \sim X_0, \lambda \in F \cup F^S \cup F^I \), for all \((Y_1, \pi_1) \) \( \in \) SUBTREE \((X_1, \theta_1)\) and \((Y_2, \pi_2) \) \( \in \) SUBTREE \((X_2, \theta_2)\) where \((X_1, \theta_1) \) \( \in \) \( t_a \) and \((X_2, \theta_2) \) \( \in \) \( t_a \), then \( n = \lambda + \pi_1 + \pi_2 \).

(b) When \( x \neq \ast \), add \((X_0, n) \) to \( t_a \) if \( f_x(Y_1, Y_2) \sim Z, \lambda \in F \cup F^I \) and \( f_x(Z_1, Z_2) \sim X_0, \lambda \in F \cup F^S \), \((Z_1, \pi_1) \) \( \in \) SUBTREE \((Z, \theta)\), \((Z_2, \pi_2) \) \( \in \) SUBTREE \((\text{Start})\), then \( n = \lambda + \pi_1 + \pi_2 \).
(c) Delete redundant states.

Step 4. If \((S, n)\) is in \(t_0\), then \(d(L(G_e), \alpha') = n\). exit

For all states in \(H_b\), a table of minimal deleted trees is first computed by the two subroutine \(\text{SUBTREE}(\text{Start})\) and \(\text{SUBTREE}(X, \theta)\). \((X_0, n)\) is generated from subroutine \(\text{SUBTREE}(\text{Start})\), if \(X_0 \xrightarrow{X} X_0 \in G_b \Rightarrow \psi\), and \(\psi\) is the smallest subtree among all the possible derivations starting from non-terminal \(X_0\). Similarly, \((X_0, n)\) is generated from subroutine \(\text{SUBTREE}(X, \theta)\), if \(X_0 \xrightarrow{X} X_0 \in G_b \Rightarrow \psi\), where \(X\) is the state of a frontier in \(\psi\), and all the other nodes in \(\psi\) are labeled by terminal symbols. Furthermore, \(\psi\) has the least number of nodes among all the possible derivations. Steps (1), (2), and (3) are formulated under the assumption that: (i) the trees represented by the states of the immediate successor of the current node may be the subtree of the actual derivation, (ii) there are one or more subtrees that may be deleted between the current node and the node right adjacent to it. For example, if rule \(X_0 \xrightarrow{X} X\) is applied at \(t_{a}\), but \((X, \theta) \in t_{a-1}\), then \((Y, \pi)\) must be in \(\text{SUBTREE}(X, \theta)\). The other words, let \(\beta\) be the tree represented by state \(X\) and \(\alpha\) be the tree represented by state \(Y\), then \(\beta\) is a subtree of \(\alpha\), where nodes in \(\alpha\) but not in \(\beta\), are assumed to be deleted, the cost of deletion is \(\pi - \theta\) which is the minimum among all the possible deletions. All the insertions are handled by transitions in \(F^I\) automatically.

Example 3.4. Recognition of Hand-Written Character E

Suppose a casually-written character E, as shown in Figure 3.23(a), whose binary tree representation \(\alpha'\) is shown in Figure 3.23(b), is to be recognized. The expanded tree automaton constructed from the tree grammar \(G_E\) in Appendix D-1 is as follows:
\( \tilde{W}_E = (\tilde{V}, \tilde{F}, \{S\}) \)
\( \tilde{V} = \{S, U_S, B, U_B, D, C, I\}, \quad \tilde{E} = \{a, b, d, h, $, *\} \)

\( \tilde{F} \):

\( \tilde{F}^D: f (B) \sim U_S, 0 \)
\( f (U_B, D) \sim B, \gamma \)
\( f (C) \sim U_B, 0 \)
\( f \sim D, \gamma \)
\( f (D) \sim C, \gamma \)

\( \tilde{F}^I: f_y (B) \sim B, \delta \)
\( f_y (D) \sim D, \delta \)
\( f_y (C) \sim C, \delta \)
\( \gamma \in \{a, b, d, h\} \)
\( f_y (I) \sim D, \sigma \)
\( \gamma \in \{a, d, h\} \)
\[ f_b (1,D) \sim C, \ 0 \]
\[ f_y (1,D) \sim C, \ \sigma \quad y \in \{a,d,h\} \]
\[ f_* (U_5, D) \sim S, \ 0 \]
\[ f_* (S, I) \sim S, \ 0 \]
\[ f_\downarrow (S, I) \sim S, \ 0 \]
\[ f_* (U_5, I) \sim U_5, \ 0 \]
\[ f_* (U_B, D) \sim B, \ 0 \]
\[ f_* (B, I) \sim B, \ 0 \]
\[ f_b (B, I) \sim B, \ 0 \]
\[ f_y (B, I) \sim B, \ \sigma \quad y \in \{a,d,h\} \]
\[ f_* (U_B, I) \sim U_B, \ 0 \]
\[ f_* (D) \sim C, \ 0 \]
\[ f_* (C, I) \sim C, \ 0 \]
\[ f_b (C, I) \sim C, \ 0 \]
\[ f_y (C, I) \sim C, \ \sigma \quad y \in \{a,d,h\} \]
\[ f_y (I) \sim I, \ \delta \quad y \in \{a,b,d,h,*\} \]
\[ f_y (I, I) \sim I, \ \delta \quad y \in \{a,b,d,h,*\} \]
\[ f_y \sim I, \ \delta \quad y \in \{a,b,d,h\} \]

Figure 3.23 Distorted Character E (a) and Its Binary Tree Representation \( \alpha^* \) (b).
Using the GECTA, the transition table of \( \alpha' \) is illustrated in Figure 3.24 (a) and (b). From Figure 3.24 (b), since \((S, \sigma + \delta + \gamma)\) is in \( t_0 \), we have \( d(L(G_E), \alpha') = \sigma + \delta + \gamma \).

### 3.4.3 An Illustrative Example on Character Recognition

A classic example, the hand-printed character recognition problem, is given in this section to illustrate the operation of GECTA. Input characters are assumed to be digitized patterns in a 16 x 16 format. After chain coding [77] and primitive extracting, input patterns are transformed into their corresponding tree representations. Grammars for sample characters are given; hence, GECTA's are constructed. Input patterns are then analyzed by the GECTA's and classified based on the minimum-distance criterion.

An example of input pattern is shown in Figure 3.25 (a). Assume the leftmost of the top row to be the starting point. From the starting point, the input pattern is chain coded point by point. The successor point is coded as A, B, ..., or H according to its relative position to the current node. The resulting chain code is shown in Figure 3.25 (b). In Figure 3.25 (b), the point labeled as "I" is the starting point. The majority code in a continuous line segment consisting of eight coded points is selected as the primitive of that line segment. Primitives selected by this method approximate the primitives defined in Figure 3.20. When a line is terminated or branched before it counts to eight points, the short line, if longer than two points, will be considered as a normal line segment; thus, a primitive. Otherwise, the line is neglected. The binary tree representation of pattern E is given in Figure 3.25 (c). The leftmost I of each strings in the column entitled
\[ (Y, \pi) \in \text{SUBTREE}(X, \theta) \]

(a) Table of Minimal Deleted Trees
(b) Transition table of $\alpha^*$

Figure 3.24 The acceptance of $\alpha^*$ by GECTA
Figure 3.25 Input Format (a), Chain Code Result (b), and Binary Tree Representation (c).
as "tree domain" corresponds to the identity of \( V \) in Definition 3.1. The rest of the symbols in the string are added according to Definition 3.2. Therefore, 1, 11, 12, 111, \ldots etc. in Figure 3.25 (c) corresponds to tree domain 0, 1, 2, 1.1, \ldots etc. by Definition 3.2.

Five characters, A, C, D, E, H, are used in this experiment. The assumed sample patterns for each of the five characters and their tree representations are shown in Figure 3.26. Their respective tree grammars are given in Appendix D.1.

A total of 26 patterns, casually or meticulously printed characters, are tested. Assume that \( n = \gamma = \delta = 1 \). The input patterns, their tree representations, as well as classification results and their distance from the assigned sample pattern are given in Appendix D.2. The entire recognition scheme is programmed in Fortran IV on a CDC 6500 computer. The cpu time used for chain coding and generating tree representation of each input is given under the title "time used for linking a tree." The actual recognition time is designated as "Time used for parsing." The average recognition time is 4.1 sec. per character.

These 26 test patterns are also processed by the (non-error-correcting) tree automata constructed respectively for grammars \( G_A, G_C, G_D, G_E, G_H \) given in Appendix D.1. Evidently, a pattern is acceptable only when its distance from a sample pattern is zero (see Appendix D.2). Therefore, only six out of the 26 test patterns are recognizable. The processing time is .012 sec. per character (average over the six recognizable patterns). Compared with their corresponding non-error-correcting schemes, error-correcting tree automata have the potential of recognizing highly noisy and distorted patterns. The trade-off is the processing efficiency.
Figure 3.26 Sample patterns of character A, C, D, E, H
Using the techniques suggested in Section 2.4.3 can certainly reduce the processing time significantly.

When a large set of training samples, including noisy and distorted patterns, is available, a pattern grammar inferred from these samples will reliably represent its corresponding pattern structure. A conventional (non-error-correcting) syntax analyzer can thus perform recognition tasks satisfactorily. On the other hand, when the number of training samples is small, a pattern grammar inferred is usually very poor in describing the actual pattern structure. A conventional syntax analyzer designed according to the inferred grammar may often reject (noisy) patterns that should be accepted. In such a case, an error-correcting syntax analyzer, such as the proposed ECTA, could certainly recognize (or accept) these (noisy) patterns by using the minimum-distance criterion.
CHAPTER 4
CLUSTERING ANALYSIS FOR SYNTACTIC PATTERNS

4.1 Introduction

In statistical pattern recognition, a pattern is represented by a vector, called a feature vector. The similarity between two patterns can often be expressed by a distance, or more generally speaking, a metric in the feature space. Cluster analysis can be performed on a set of patterns on the basis of a selected similarity measure [3]. In syntactic or linguistic pattern recognition [5], a pattern is represented by a sentence in a language. The sentence could be a string, a tree, or a graph of pattern primitives and relations. The emphasis of such a representation is on the structure of patterns which is described by the syntax of a language. A similarity measure between two syntactic patterns must include the similarity of both their structures and primitives.

In Chapter 2 and Chapter 3, we have proposed distance measures for strings and trees, which leads to the study of clustering analysis for syntactic patterns.

The conventional clustering methods, such as, the minimum spanning tree, the nearest (or K-nearest) neighbor classification rule and the method of clustering centers can be extended to syntactic patterns. We shall briefly describe the extension in Section 4.2. An illustrative example using a set of character patterns are presented in Section 4.3.

The studies described in Section 4.2 and Section 4.3 are mainly on the pattern-to-pattern basis. An input sentence (a pattern) is
compared with sentences in a formed cluster, one by one, or with the representation (cluster center) of the cluster. In Section 4.5 we shall use the distance measure between a sentence and a language proposed in Chapter 2. The proposed clustering procedure is combined with a grammatical inference procedure and an error-correcting parsing technique. The idea is to model the formed cluster by inferring a grammar, which implicitly characterize the structural identity of the cluster. The language generated by the grammar may be larger than the set consisting of the members of the cluster, and includes some possible similar patterns due to the recursive nature of grammar. Then the distance between an input sentence (a syntactic pattern) and a language (a group of syntactic pattern) is computed by using a ECP (error-correcting parser). The recognition is based on the nearest neighbor rule.

4.2. Sentence-to-Sentence Clustering Algorithms

4.2.1. A Nearest Neighbor Classification Rule

Suppose that \( C_1 \) and \( C_2 \) are two pattern sets, represented by sentences \( X_1 = \{x_1^1, x_2^1, \ldots, x_n^1\} \) and \( X_2 = \{x_1^2, x_2^2, \ldots, x_n^2\} \) respectively. For an unknown syntactic pattern \( y \), decide that \( y \) is in the same class as \( C_1 \) if

\[
\min_{j} d(x_j^1, y) < \min_{\ell} d(x_\ell^2, y),
\]

and \( y \) is in class \( C_2 \) if

\[
\min_{j} d(x_j^1, y) > \min_{\ell} d(x_\ell^2, y).
\]  \( \text{(4.1)} \)

In order to determine \( \min_{j} d(x_j^1, y) \), for some \( i \), the distance between \( y \) and every element in the set \( X_1 \) have to be computed individually. The
string-to-string correction algorithm proposed in [26] yields exactly the distance between two strings defined in Definition 2.4. We shall briefly describe the algorithm in Appendix G.1.

The nearest neighbor classification rule can be easily extended to the K-nearest neighbor rule. Let \( \tilde{X}_j = \{ \tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_{n_1} \} \) be a reordered set of \( X_j \) such that \( d(\tilde{x}_j, y) \leq d(\tilde{x}_{j'}', y) \) if \( j < j' \), for all \( 1 \leq j, j' \leq n_1 \), then

\[
\text{decide } y \in \begin{cases} 
C_1 & \text{if } \sum_{j=1}^{K} \frac{1}{K} d(\tilde{x}_j, y) \leq \sum_{j=1}^{K} \frac{1}{K} d(\tilde{x}_j', y) \\
C_2 & \text{otherwise}
\end{cases}
\]  

(4.2)

We shall describe a clustering procedure in which, the classification of an input pattern is based on the nearest (or K-nearest) neighbor rule.

Algorithm 4.1

Input: A set of samples \( X = \{x_1, x_2, \ldots, x_n\} \) and a design parameter, or threshold, \( t \).

Output: The partition of \( X \) into \( m \) clusters, \( C_1, C_2, \ldots, C_m \).

Method:

Step 1. Assign \( x_i \) to \( C_j \), \( j=1, m=1 \).

Step 2. Increase \( j \) by one. If \( D = \min_{i} d(x_i, x_j) \) is the minimum, \( 1 \leq i \leq m \), and

(1) \( D \leq t \), then assign \( x_j \) to \( C_i \)

(11) \( D > t \), then initiate a new cluster for \( x_j \), and increase \( m \) by one.

Step 3. Repeat Step 2 until all the elements of \( X \) have been put in a cluster.
Note that, in Algorithm 4.1, a design parameter is required. A commonly used clustering procedure is to construct a minimum spanning tree. Each node on the minimum spanning tree represents an element in the sample set $X$. Then partition the tree. Actually, when the distances between all of the pairs, $d(x_i, x_j)$, $x_i, x_j \in X$, are available, the algorithm for constructing minimum spanning tree is the same as that where $X$ is a set of feature vectors in the statistical pattern recognition [94]. The algorithm is given in Appendix G.2.

### 4.2.2. The Cluster Center Techniques

Let us define a $\beta$-metric for a sentence $x_j \mid$ in cluster $C_i$ as follow,

$$
\beta_j \mid = \frac{1}{n_i} \sum_{i=1}^{n_i} d(x_j \mid, x_i \mid)
$$

(4.3)

Then, $x_j \mid$ is the cluster center of $C_i$, if $\beta_j \mid = \min \{\beta_{\ell} \mid \mid 1 \leq \ell \leq n_i\}$, $x_j \mid$ is also called the representation of $C_i$, denoted $A_i$.

The following clustering algorithm is given in [94].

**Algorithm 4.2**

*Input:* A sample set $X = \{x_1, x_2, \ldots, x_n\}$.

*Output:* The partition of $X$ into $m$ cluster

*Method:*

**Step 1.** Let $m$ elements of $X$, chosen at random, be the "representation" of the $m$ clusters. Let them be called $A_1, A_2, \ldots, A_m$.

**Step 2.** For all $i$, $x_i \in X$ is assigned to cluster $j$, iff $d(A_j, x_i)$ is minimum.

**Step 3.** For all $j$, a new mean $A_j$ is computed. $A_j$ is the new representation of cluster $j$.

**Step 4.** If no $A_j$ has changed, stop. Otherwise, go to Step 2.
4.3. An Illustrative Example

4.3.1. Data Preparation

A set of English characters is used to illustrate the proposed clustering procedure. There are 51 characters from nine different classes: D, F, H, K, P, U, V, X, and Y. Eight of the nine classes are selected from four groups, each with characters of similar structure; that is, D and P, H and K, U and V, and X and Y. The class of character F is different from the other eight classes. Each character is a continuous line pattern on a 20 x 20 grid as shown in Figure 4.1(a). Starting from its lower left corner, each input pattern is chain-coded cell by cell by a subroutine. Each successive cell is coded as A, B, C, or D according to its position relative to that of the current one. After three consecutive cells have been coded, a pattern primitive of this line segment is extracted.

(a) Primitive and Subpattern Extraction

Four pattern primitives which are line segments with four different orientations, \(a, b, c, d\), are selected. For example, ABA, CCA, or ADD are reduced to primitive a, b, or d, respectively. A primitive extraction subroutine PRIMITIVE is constructed and several typical results from the subroutine are shown in Figure 4.2. In the meantime, the chain-code subroutine searches for singular points of the input pattern for segmentation purposes. Coordinates of the singular points at two ends of a line segment (called a branch) are then recorded. For example, the chain-coded result for the character P shown in Figure 4.1(a) is given in Figure 4.1(b), in which the symbol "S" represents singular points. From Figure 4.1(b), the pattern has three branches, which,
Figure 4.1 The primitive extraction of a character P
Figure 4.2 The primitive selection of a line segment
together with the coordinates of their end points, are shown in Figure 4.1(c). Each branch is a subpattern, and the three branches consist of primitive strings bbb (Branch 1), b (Branch 2), and ddbbd (Branch 3), respectively.

(b) String Representation

From the recorded coordinates of starting and terminating points (tail and head) of each branch, the subroutine CONCATATE determines concatenation relations between branches. Following Shaw's PDL [95], three concatenation relations, +, x, *, and the parentheses ( and ) are used. However, * is used here primarily for the situation of a "self loop"; that is, a branch of which the head and the tail coincide. In such a case, the notation (B)* is used where B is the branch. The priority of the three concatenation relations follows the order of *, x, and then +. Consequently, redundant parentheses are eliminated. For the character P in Figure 4.1(a), the concatenations of branches are expressed as \( \text{BRANCH 1} + (\text{BRANCH 2} + \text{BRANCH 3})^* \). The final string representation in terms of pattern primitives is \( \text{bbb} + (\text{b} + \text{dddbd})^* \). Similarly, the character K in Figure 4.3 is represented by \( \text{BRANCH 1} + \text{BRANCH 2} \times \text{BRANCH 3} \times \text{BRANCH 4} \), and finally by a string of primitives \( \text{b} + \text{bbbxaaxbc} \).

The 51 chain-coded patterns with their pattern numbers which represent input sequence are shown in Figure 4.4. The string representations extracted from subroutines, CHAIN-CODE, PRIMITIVE and CONCATATE, are also listed in Table 4.1.

4.3.2. Experiments

(a) The Distance

The proposed distance measure is performed on the linguistic representations of pattern (sentences), rather than on the pattern
Figure 4.3 The primitive extraction of a character K
Figure 4.4 The 51 chain-coded character patterns
Figure 4.4 continued
Figure 4.4 continued
Figure 4.4 continued
<table>
<thead>
<tr>
<th>Pattern No.</th>
<th>String Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>bbb+(b+dddbdd)*</td>
</tr>
<tr>
<td>2</td>
<td>aa+cddxaaxcb</td>
</tr>
<tr>
<td>3</td>
<td>(bbb+ddcbbaa)*</td>
</tr>
<tr>
<td>4</td>
<td>cbbxdaabbb</td>
</tr>
<tr>
<td>5</td>
<td>bb+(b+dd)xd</td>
</tr>
<tr>
<td>6</td>
<td>bbb+ccxbb</td>
</tr>
<tr>
<td>7</td>
<td>(bbb+dxddcbbaa)*</td>
</tr>
<tr>
<td>8</td>
<td>ba+bbxaaxcc</td>
</tr>
<tr>
<td>9</td>
<td>aac+cxaaxb</td>
</tr>
<tr>
<td>10</td>
<td>b+bbxdddd+bxb</td>
</tr>
<tr>
<td>11</td>
<td>(bbb+dxddbad)*</td>
</tr>
<tr>
<td>12</td>
<td>bb+(bb+d d)xd</td>
</tr>
<tr>
<td>13</td>
<td>cbbxaabbba</td>
</tr>
<tr>
<td>14</td>
<td>bb+bbxaaxcc</td>
</tr>
<tr>
<td>15</td>
<td>bb+ccxa</td>
</tr>
<tr>
<td>16</td>
<td>cbbxda+bbxb</td>
</tr>
<tr>
<td>17</td>
<td>bb+bbxaaxbccc</td>
</tr>
<tr>
<td>18</td>
<td>bb+(bb+dddbaa)*</td>
</tr>
<tr>
<td>19</td>
<td>b+bbxbd+bxdb</td>
</tr>
<tr>
<td>20</td>
<td>aa+cbxaaxcc</td>
</tr>
<tr>
<td>21</td>
<td>bbb+(bbadcbad)*</td>
</tr>
<tr>
<td>22</td>
<td>bbbxdabbb</td>
</tr>
<tr>
<td>23</td>
<td>bbb+bcxea</td>
</tr>
<tr>
<td>24</td>
<td>b+(bb+ddcbad)*</td>
</tr>
<tr>
<td>25</td>
<td>b+bbbxaxbc</td>
</tr>
<tr>
<td>26</td>
<td>bbbbxabaa</td>
</tr>
<tr>
<td>27</td>
<td>bbb+(dx(b+dd))xddd</td>
</tr>
<tr>
<td>28</td>
<td>bb+cbxaaxa</td>
</tr>
<tr>
<td>29</td>
<td>bb+bbxddd+axbb</td>
</tr>
<tr>
<td>30</td>
<td>baa+bbxdaaxcc</td>
</tr>
<tr>
<td>31</td>
<td>ba+abxdd+axbbb</td>
</tr>
<tr>
<td>32</td>
<td>aa+ccxaxcc</td>
</tr>
<tr>
<td>33</td>
<td>bb+(bb+dxddcaad)*</td>
</tr>
<tr>
<td>34</td>
<td>bb+bbxa+axcb</td>
</tr>
<tr>
<td>35</td>
<td>bb+(bb+d)xd</td>
</tr>
<tr>
<td>36</td>
<td>cbbxdaabb</td>
</tr>
<tr>
<td>37</td>
<td>ba+bcxaxa</td>
</tr>
<tr>
<td>38</td>
<td>aa+ccxaaxcc</td>
</tr>
<tr>
<td>39</td>
<td>bb+(bb+dd)xd</td>
</tr>
<tr>
<td>40</td>
<td>(bbbb+ddcbbaa)*</td>
</tr>
<tr>
<td>41</td>
<td>dab+cbxba</td>
</tr>
<tr>
<td>42</td>
<td>bb+(b+dd)xd</td>
</tr>
<tr>
<td>43</td>
<td>bbb+ccxa</td>
</tr>
<tr>
<td>44</td>
<td>b+dxabxddd+bxb</td>
</tr>
<tr>
<td>45</td>
<td>(bb+ddcbdd)*</td>
</tr>
<tr>
<td>46</td>
<td>aac+cxaaxcc</td>
</tr>
<tr>
<td>47</td>
<td>bbb+ddaabcddd</td>
</tr>
<tr>
<td>48</td>
<td>bbbxda+bbxb</td>
</tr>
<tr>
<td>49</td>
<td>baa+ccxbxbxccc</td>
</tr>
<tr>
<td>50</td>
<td>ba+bx+abxbbbb</td>
</tr>
<tr>
<td>51</td>
<td>cbbxdabbbb</td>
</tr>
</tbody>
</table>

Table 4.1 The 51 character patterns and their representations
themselves. Consequently, whether the model yields a good measurement is a matter of choosing representations. Figure 4.5 illustrates a couple of examples. In Figure 4.5(a) the distance between the two K's is smaller than a K and an X. Similarly, in Figure 4.5(b) the distance between a U and a distorted U, 'U\prime', is still smaller than the distorted U and a H.*

(b) **A Minimum Spanning Tree**

Using algorithm in Appendix G.2 the minimum spanning tree for the 51 characters is constructed and shown in Figure 4.6. The true clusters are circled on the tree.

(c) **The Clustered Results Using Algorithm 4.1**

The results of clustering based on K-nearest neighbor classification rule are given in Figure 4.7, Figure 4.8 and Figure 4.9 when K=1 and t=6, K=3 and t=6, K=3 and t=6.5 respectively, where t is the preset threshold. The case that K=1 is the same as that using the nearest neighbor rule.

(d) **The Clustering Results by Computing Cluster Centers**.

The clustering procedure given in Algorithm 4.2 does not require a preset parameter. Let the algorithm be initiated by choosing the first nine input patterns; patterns 1 to 9 as the representations of the 9 classes. The procedure becomes stable after three iterations. The results of all the iterations are given in Table 4.2. The final clusters are shown in Figure 4.10. The representation of each formed cluster is marked with a square.

*In this section, all the experiments use unweighted distance. However, we exclude the possibility that a primitive namely, a, b, c, d, can not substitute a relation symbol, namely, +, x, *, (, ), and vice versa.*
Figure 4.5 The distances between similar and dissimilar patterns.
Figure 4.5 Continued

Pattern 16

Pattern 10

Pattern 36

Pattern 29

\[ \text{Pattern } 16 \]

\[ \text{Pattern } 10 \]

\[ \text{Pattern } 36 \]

\[ \text{Pattern } 29 \]
Figure 4.6 The minimum spanning tree of the 51 character patterns
Figure 4.7 The result of using K-nearest neighbor recognition rule - K=1, t=6
Figure 4.8 The result of using K-nearest neighbor recognition rule - $K=3$, $t=6$
Figure 4.9 The result of using K-nearest neighbor recognition rule – K=3, t=6.5
<table>
<thead>
<tr>
<th>cluster</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>21</td>
<td>18</td>
<td>10</td>
<td>12</td>
<td>15</td>
<td>11</td>
<td>14</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>second</td>
<td>33</td>
<td>24</td>
<td>13</td>
<td>27</td>
<td>19</td>
<td>17</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>40</td>
<td>16</td>
<td>35</td>
<td>23</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>22</td>
<td>39</td>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>42</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31</td>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(a) the first iteration

<table>
<thead>
<tr>
<th>cluster</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>1</td>
<td>9</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>14</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>second</td>
<td>24</td>
<td>18</td>
<td>10</td>
<td>12</td>
<td>23</td>
<td>17</td>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>40</td>
<td>13</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>49</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>45</td>
<td>16</td>
<td>35</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>42</td>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) the second iteration

Table 4.2 The three iteration results using Algorithm 4.2
Table 4.2 Continued

<table>
<thead>
<tr>
<th>cluster</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>representation</td>
<td>24</td>
<td>9</td>
<td>3</td>
<td>22</td>
<td>39</td>
<td>15</td>
<td>11</td>
<td>8</td>
<td>38</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>14</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>46</td>
<td>18</td>
<td>10</td>
<td>12</td>
<td>23</td>
<td>17</td>
<td>49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>40</td>
<td>13</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>45</td>
<td>16</td>
<td>35</td>
<td>28</td>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>42</td>
<td>37</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>41</td>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>43</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) the third iteration
Figure 4.10 The result of classification according to clustering centers
(e) **Remark**

From the experiments described in (d) and their results, the main weaknesses that exist in conventional clustering algorithms are also unavoidable when data are in the form of sentences. These weaknesses are; (1) the requirement of thresholds of somewhat arbitrary nature, (2) the requirement of large memory, for example, tables of \( n(n-1)/2 \) if the sample set, \( X \), has \( n \) objects, and (3) the chain effect. However, the chain effect could be improved by using a weighted distance.

4.4. **A Proposed Nearest Neighbor Syntactic Recognition Rule**

With the distance between a sentence and a language defined in Chapter 2, we can construct a syntactic recognizer using the nearest (or K-nearest) neighbor rule. Suppose that we are given two classes of patterns characterized by grammar \( G_1 \) and \( G_2 \), respectively. For an unknown syntactic pattern \( y \), decide that \( y \) is in the same class as \( L(G_1) \) if

\[
d(L(G_1), y) > d(L(G_2), y)
\]

and decide that \( y \) is in the same class as \( L(G_2) \) if,

\[
d(L(G_2), y) > d(L(G_1), y)
\]

(4.4)

The distance \( d(L(G_1), y) \) can be determined by a minimum-distance ECP constructed for \( G_1 \). Consequently, a grammatical inference procedure is required to infer a grammar for each class of pattern samples. Since the parser also gives the structural description of \( y \), the syntactic recognizer gives both the classification and description of \( y \) as its output. We shall summarize the procedure in the following algorithm.
Algorithm 4.3

Input: \( m \) sets of syntactic pattern samples
\[ X_1 = \{x_1^1, x_2^1, \ldots, x_n^1\}, \ldots, X_m = \{x_1^m, x_2^m, \ldots, x_n^m\} \]
and a pattern \( y \) with unknown classification.

Output: The classification and structural description of \( y \).

Method:

Step 1. Infer \( m \) grammars \( G_1, G_2, \ldots, G_m \), from \( X_1, X_2, \ldots, X_m \), respectively.

Step 2. Construct minimum-distance ECP's, \( E_1, E_2, \ldots, E_m \) for
\( G_1, G_2, \ldots, G_m \), respectively.

Step 3. Calculate \( d(L(G_k), y) \) for all \( k = 1, \ldots, m \). Determine \( \ell \) such that
\[ d(L(G_\ell), y) = \min_k d(L(G_k), y) \]

\( y \) is then classified as class \( \ell \). In the meantime, the structural description of \( x \) can be obtained from \( E_\ell \).

4.5. A Clustering Procedure for Syntactic Patterns

4.5.1. The Algorithm

Using the distance defined in Chapter 2 as a similarity measure between a syntactic pattern and a set of syntactic patterns, we can perform a cluster analysis to syntactic patterns. The procedure again involves error-correcting parsing and grammatical inference. In contrast to the nearest neighbor rule in Section 4.4 which uses a supervised inference procedure, the procedure described in this section is basically non-supervised. When the syntactic pattern samples are observed sequentially, a grammar can be easily inferred for the sample observed at each stage of the clustering procedure. We propose the following clustering procedure for syntactic patterns:
Algorithm 4.4

Input: A set of syntactic pattern samples $X = \{x_1, x_2, ..., x_n\}$ where $x_i$ is a string of terminals or primitives. A threshold $t$.

Output: The assignment of $x_i$, $i=1,...,n$ to $m$ clusters and the grammar $G^{(k)}$, $k=1,...,m$, characterizing each cluster.

Method:

Step 1. Input the first sample $x_1$, infer a grammar $G_1^{(1)}$ from $x_1$, $L(G_1^{(1)}) \ni \{x_1\}$.

Step 2. Construct an error-correcting parser $E_1^{(1)}$ for $G_1^{(1)}$.

Step 3. Input the second sample $x_2$, use $E_1^{(1)}$ to determine whether or not $x_2$ is similar to $x_1$ by comparing the distance between $L(G_1^{(1)})$ and $x_2$, i.e., $d(x_2, L(G_1^{(1)}))$, with a threshold $t$.

1. If $d(x_2, L(G_1^{(1)})) < t$, $x_1$ and $x_2$ are put into the same cluster (Cluster 1). Infer a grammar $G_2^{(1)}$ from $\{x_1, x_2\}$.

2. If $d(x_2, L(G_1^{(1)})) \geq t$, initiate a new cluster for $x_2$ (Cluster 2) and infer a new grammar $G_1^{(2)}$ from $x_2$. In this case, there are two clusters characterized by $G_1^{(1)}$ and $G_1^{(2)}$, respectively.

Step 4. Repeat Step 2, construct error-correcting parsers for $G_2^{(1)}$ or $G_1^{(2)}$ depending upon $d(x_2, L(G_1^{(1)})) < t$ or $d(x_2, L(G_1^{(1)})) \geq t$, respectively.

Step 5. Repeat Step 3 for a new sample. Until all the pattern samples are observed, we have $m$ clusters characterized by $G_1^{(1)}$, $G_2^{(2)}$, ..., $G_m^{(m)}$, respectively.
The parsers (non-error-correcting) constructed according to 
\(G_{n_1}^{(1)}, G_{n_2}^{(2)}, \ldots, G_{n_m}^{(m)}\) could then form a syntactic recognizer directly 
for the m-class recognition problem.

The threshold \(t\) is a design parameter. It can be determined from 
a set of pattern samples with known classifications. For example, if 
we know that the sample \(x_i\) is from Class 1 characterized by \(G^{(1)}\) and the 
sample \(x_j\) is from Class 2 characterized by \(G^{(2)}\), then \(t < d(x_i, x_j)\). 
Or, more generally speaking,

\[
t < \min \{d(L(G^{(2)}), x_i), d(L(G^{(1)}), x_j)\}
\]

(4.5)

For \(m\) classes characterized by \(G^{(1)}, G^{(2)}, \ldots, G^{(m)}\), respectively, we can 
choose

\[
t < \min \{d(L(G^{(k)}), x^{(k)})\}, k \neq \ell
\]

(4.6)

where \(x^{(k)}\) is a pattern sample known from Class \(k\) and \(L(G^{(\ell)})\) is the 
grahcr characterizing Class \(\ell\) (\(\ell \neq k\)). If the above required information 
is not available, an appropriate value of \(t\) will have to be determined 
on an experimental basis until a certain stopping criterion is satisfied 
(for example, with a known number of clusters).

4.5.2 An Experiment

Let the same set of pattern samples used in Section 4.3 be tested 
by Algorithm 4.4. The subroutines of finding string representations 
for the 51 character patterns are still used here. In addition, we will 
have subroutines for grammatical inference and error-correcting parsing.
(1) Grammatical Inference (Step 1, 3, and 5 in Algorithm 4.4)

By comparing its distances to existing clusters with a threshold \( t \), an input pattern sample is assigned to an existing cluster or a new cluster. In either case, a grammar is first inferred for the single input sample. The inferred grammar is then merged into the grammar (by merging their productions) characterizing the assigned cluster. The subroutine REDUCE combines productions of the two grammars, removes identical productions, and unifies nonterminals. To use a simple inference procedure, input samples are assumed to be generated by finite-state grammars. However, non-self-embedding context-free productions are used to describe concatenation of branches. Each branch is described by finite-state productions. For the character \( P \) shown in Figure 4.1, the inferred grammar is given in Table 4.3.

(2) Error-Correcting Parser (ECP - Step 2 and 4)

The error-correcting parser used in this example is the MDECP given in Algorithm 2.1 and 2.2. Certain realistic assumptions are made to reduce the number of error productions. For example, we do not allow a substitution error that occurred between \( a, b, c, \) or \( d \) and \( +, x, * \), \( (,or) \). Also, the concatenation symbol \( + \) or \( x \) cannot be inserted at the end of a string.

A simplified flow chart for the complete experiment is given in Figure 4.11.

(3) Experimental Results

Following the clustering procedure described in Section 4.5, three experiments were performed: (i) using unweighted (Levenshtein) distance with threshold \( t = 3 \), (ii) using unweighted (Levenshtein) distance with
\( G_1^{(0)} = (V_N, V_T, P, S_0) \)

\( V_N = \{ S_0, BA, KA, BC, EA, EB \} \)

\( V_T = \{ (, +, \times, *, ), b, d \} \)

\[
P: \quad
S_0 \rightarrow BA + KA *
KA \rightarrow ( BA + BC )
BA \rightarrow b BA
BA \rightarrow b
BC \rightarrow d BC
BC \rightarrow d EA
EA \rightarrow b EB
EB \rightarrow d EB
EB \rightarrow d
\]

\( L(G_1^{(0)}) = \{ b^{n_1} + (b^{n_2} + d^{n_3} b d^{n_4})* | n_1, n_2, n_3, n_4 \text{ are positive integers} \}. \)

Table 4.3 An inferred grammar for pattern 1.
n: number of pattern samples
i: input sequence
m: current number of clusters
j: the closest cluster for input i
l: assigned cluster for input i
D: D = d(x_i, L(G_{i-1}(j)))
t: threshold

Figure 4.11 Flow chart of the clustering Algorithm 4.4
threshold \( t = 4 \), and (iii) using weighted distance with threshold \( t = 3 \). The results of clustering analysis are listed in Table 4.4. In Table 4.4 pattern samples are grouped according to the clustering result of the experiment (iii); pattern numbers in the first column correspond to those listed in Figure 4.4. The number in the parentheses of the last column in Table 4.4 represents the weighted distance between the pattern and the most similar cluster; that is, the cluster to which the pattern is assigned.

For Experiment (i), there were 11 clusters formed. Pattern 18 was not assigned to the same cluster with other character P's, and Pattern 27 was not assigned to the same cluster with other F's. For Experiment (ii), \( t \) was increased from 3 to 4, only 7 clusters were formed. Both Pattern 18 and Pattern 27 were correctly clustered. However, all the X's and the K's were assigned to the same cluster. In both experiments, Patterns 13, 26 and 47 were not correctly clustered. For Experiment (iii), 10 clusters were formed; all the patterns except Pattern 47 were correctly clustered. The weights associated with each error productions used in the experiment are given in Table 4.5.

The final inferred grammar \( G_n, n = 51 \), from the third experiment, is listed in Table 4.6. The grammar is the union of the inferred grammars for each cluster; namely, \( G^{(0)}, G^{(1)}, G^{(2)}, \ldots, G^{(9)} \). Nonterminals \( S_0, S_1, S_2, \ldots, S_9 \) in the grammar are the start symbols of \( G^{(0)}, G^{(1)}, G^{(2)}, \ldots, G^{(9)} \), respectively. In order to carry out the parsing for all clusters using a single parser, we merge all the productions originated from \( S_0, S_1, \ldots, S_9 \) into a single grammar \( G_{51} \) by adding productions \( SS \rightarrow S_0, SS \rightarrow S_1, \ldots, SS \rightarrow S_9 \) where \( SS \) is the start symbol of \( G_{51} \).
<table>
<thead>
<tr>
<th>Pattern No.</th>
<th>String Representation</th>
<th>True Character</th>
<th>Unweighted</th>
<th>Weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>t=3</td>
<td>t=4</td>
</tr>
<tr>
<td>1</td>
<td>bbb+(b+ddbd)*</td>
<td>P</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>bbb+(bb+ddbd)*</td>
<td>P</td>
<td>0</td>
<td>0(2.2)</td>
</tr>
<tr>
<td>21</td>
<td>bbb+(bbadcbad)*</td>
<td>P</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>b+(bb+ddcbad)*</td>
<td>P</td>
<td>0</td>
<td>0(1.5)</td>
</tr>
<tr>
<td>33</td>
<td>bb+(bb+dxddcaad)*</td>
<td>P</td>
<td>0</td>
<td>0(1.5)</td>
</tr>
<tr>
<td>2</td>
<td>aaaa+cddxaaxcb</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>aax+ccxxxc</td>
<td>X</td>
<td>1</td>
<td>1(1.5)</td>
</tr>
<tr>
<td>20</td>
<td>aa+cbxaaxcc</td>
<td>X</td>
<td>1</td>
<td>1(0.6)</td>
</tr>
<tr>
<td>32</td>
<td>aa+cxaxxb</td>
<td>X</td>
<td>1</td>
<td>1(1.0)</td>
</tr>
<tr>
<td>38</td>
<td>aa+ccxaxxcc</td>
<td>X</td>
<td>1</td>
<td>1(0.0)</td>
</tr>
<tr>
<td>46</td>
<td>aa+cxaxxcc</td>
<td>X</td>
<td>1</td>
<td>1(0.0)</td>
</tr>
<tr>
<td>49</td>
<td>baac+ccxbbxcc</td>
<td>X</td>
<td>1</td>
<td>1(2.7)</td>
</tr>
<tr>
<td>3</td>
<td>(bb+ddcbdd)*</td>
<td>D</td>
<td>2</td>
<td>2(2.1)</td>
</tr>
<tr>
<td>7</td>
<td>(bb+dxddcbdd)*</td>
<td>D</td>
<td>2</td>
<td>2(2.1)</td>
</tr>
<tr>
<td>11</td>
<td>(bb+dxddbbad)*</td>
<td>D</td>
<td>2</td>
<td>2(2.1)</td>
</tr>
<tr>
<td>40</td>
<td>(bb+dxddcbad)*</td>
<td>D</td>
<td>2</td>
<td>2(2.1)</td>
</tr>
<tr>
<td>45</td>
<td>(bb+ddcbdd)*</td>
<td>D</td>
<td>2</td>
<td>2(0.9)</td>
</tr>
<tr>
<td>4</td>
<td>cbb+xdabbb</td>
<td>U</td>
<td>3</td>
<td>3(3.0)</td>
</tr>
<tr>
<td>16</td>
<td>cbb+adab+bbxb</td>
<td>U</td>
<td>3</td>
<td>3(2.0)</td>
</tr>
<tr>
<td>22</td>
<td>bbxdabbb</td>
<td>U</td>
<td>3</td>
<td>3(0.5)</td>
</tr>
<tr>
<td>36</td>
<td>cbb+adabbb</td>
<td>U</td>
<td>3</td>
<td>3(0.0)</td>
</tr>
<tr>
<td>48</td>
<td>bbxdabbb</td>
<td>U</td>
<td>3</td>
<td>3(0.5)</td>
</tr>
<tr>
<td>51</td>
<td>cbb+adabbb</td>
<td>U</td>
<td>3</td>
<td>3(0.0)</td>
</tr>
<tr>
<td>5</td>
<td>b+b+dd)+xd</td>
<td>F</td>
<td>4</td>
<td>4(4.0)</td>
</tr>
<tr>
<td>12</td>
<td>b+b+(bb+dd)+xd</td>
<td>F</td>
<td>4</td>
<td>4(4.0)</td>
</tr>
<tr>
<td>27</td>
<td>bbb+(dx(b+dd)+xd)</td>
<td>F</td>
<td>9</td>
<td>4(2.0)</td>
</tr>
<tr>
<td>35</td>
<td>b+b+(bb+dd)+xd</td>
<td>F</td>
<td>4</td>
<td>4(4.0)</td>
</tr>
<tr>
<td>39</td>
<td>b+b+(bb+dd)+xd</td>
<td>F</td>
<td>4</td>
<td>4(4.0)</td>
</tr>
<tr>
<td>42</td>
<td>b+b+(bb+dd)+xd</td>
<td>F</td>
<td>4</td>
<td>4(4.0)</td>
</tr>
<tr>
<td>6</td>
<td>bbb+ccxb</td>
<td>Y</td>
<td>5</td>
<td>5(5.0)</td>
</tr>
<tr>
<td>15</td>
<td>bbb+ccxx</td>
<td>Y</td>
<td>5</td>
<td>5(2.0)</td>
</tr>
<tr>
<td>23</td>
<td>bbb+bcxx</td>
<td>Y</td>
<td>5</td>
<td>5(0.9)</td>
</tr>
<tr>
<td>28</td>
<td>b+cbxxx</td>
<td>Y</td>
<td>5</td>
<td>5(1.0)</td>
</tr>
<tr>
<td>37</td>
<td>ba+bcxx</td>
<td>Y</td>
<td>5</td>
<td>5(1.0)</td>
</tr>
<tr>
<td>41</td>
<td>d+b+cxx</td>
<td>Y</td>
<td>5</td>
<td>5(2.9)</td>
</tr>
<tr>
<td>43</td>
<td>bbb+cxx</td>
<td>Y</td>
<td>5</td>
<td>5(0.9)</td>
</tr>
<tr>
<td>8</td>
<td>bb+bbxaxxc</td>
<td>K</td>
<td>6</td>
<td>1(6.0)</td>
</tr>
<tr>
<td>14</td>
<td>bb+bbxaxxc</td>
<td>K</td>
<td>6</td>
<td>1(6.0)</td>
</tr>
<tr>
<td>17</td>
<td>bb+bbxaxxc</td>
<td>K</td>
<td>6</td>
<td>1(6.0)</td>
</tr>
<tr>
<td>25</td>
<td>b+bbxaxxc</td>
<td>K</td>
<td>6</td>
<td>1(6.0)</td>
</tr>
<tr>
<td>30</td>
<td>baa+bbxaxxc</td>
<td>K</td>
<td>6</td>
<td>1(6.0)</td>
</tr>
<tr>
<td>34</td>
<td>bbb+bbxaxxc</td>
<td>K</td>
<td>6</td>
<td>1(6.0)</td>
</tr>
<tr>
<td>10</td>
<td>b+bbxxdd+bbxb</td>
<td>H</td>
<td>7</td>
<td>6(7.0)</td>
</tr>
</tbody>
</table>

Table 4.4 The result clusters of the 51 characters
<table>
<thead>
<tr>
<th>Pattern No.</th>
<th>String Representation</th>
<th>True Character</th>
<th>Unweighted $t=3$</th>
<th>Unweighted $t=4$</th>
<th>Weighted $t=3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>$b+b+bbx+dx+xb$</td>
<td>$H$</td>
<td>7</td>
<td>6</td>
<td>7 (1.5)</td>
</tr>
<tr>
<td>29</td>
<td>$bb+bb+bb+dx+dd+ax+bb$</td>
<td>$H$</td>
<td>7</td>
<td>6</td>
<td>7 (1.0)</td>
</tr>
<tr>
<td>31</td>
<td>$ba+ab+dx+dd+ax+bb$</td>
<td>$H$</td>
<td>7</td>
<td>6</td>
<td>7 (2.0)</td>
</tr>
<tr>
<td>44</td>
<td>$b+dx+ab+dx+dd+bx+bb$</td>
<td>$H$</td>
<td>7</td>
<td>6</td>
<td>7 (3.0)</td>
</tr>
<tr>
<td>50</td>
<td>$ba+b+xa+ab+xb+bb$</td>
<td>$H$</td>
<td>7</td>
<td>6</td>
<td>7 (3.0)</td>
</tr>
<tr>
<td>57</td>
<td>$c+bb+xa+b+bb+ba$</td>
<td>$V$</td>
<td>3</td>
<td>3</td>
<td>8 (-)</td>
</tr>
<tr>
<td>47</td>
<td>$c+bb+b+xd+a+b+bb$</td>
<td>$V$</td>
<td>5</td>
<td>5</td>
<td>8 (0.5)</td>
</tr>
<tr>
<td>47</td>
<td>$c+bb+b+xd+a+b+bb$</td>
<td>$D$</td>
<td>10</td>
<td>7</td>
<td>9 (-)</td>
</tr>
<tr>
<td>Rule</td>
<td>Weight</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>--------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_a + a</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_a + b</td>
<td>.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_a + c</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_a + d</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_a + aE_a</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_a + dE_a</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_a + +E_a</td>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_a + xE_a</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_a + λ</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + b</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + a</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + c</td>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + d</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + aE_b</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + bE_b</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + bE_b</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + +E_b</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + xE_b</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_b + λ</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_c + c</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_c + b</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_c + bE_c</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_c + cE_c</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_c + λ</td>
<td>0.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_d + d</td>
<td>0.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_d + a</td>
<td>1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E_d + b</td>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5 New rules added to the original grammars
Table 4.5 Continued

<table>
<thead>
<tr>
<th>Rule</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_d \rightarrow c$</td>
<td>0.5</td>
</tr>
<tr>
<td>$E_d \rightarrow dE_d$</td>
<td>0.0</td>
</tr>
<tr>
<td>$E_d \rightarrow xE_d$</td>
<td>1.0</td>
</tr>
<tr>
<td>$E_d \rightarrow \lambda$</td>
<td>2.1</td>
</tr>
<tr>
<td>$E_x \rightarrow x$</td>
<td>0.0</td>
</tr>
<tr>
<td>$E_x \rightarrow dE_x$</td>
<td>1.0</td>
</tr>
<tr>
<td>$E_x \rightarrow xE_x$</td>
<td>0.5</td>
</tr>
<tr>
<td>$E_+ \rightarrow +$</td>
<td>0.0</td>
</tr>
<tr>
<td>$E_+ \rightarrow aE_+$</td>
<td>1.0</td>
</tr>
<tr>
<td>$E_+ \rightarrow \lambda$</td>
<td>0.5</td>
</tr>
<tr>
<td>$E_* \rightarrow *$</td>
<td>0.0</td>
</tr>
<tr>
<td>$E) \rightarrow )$</td>
<td>0.0</td>
</tr>
<tr>
<td>$E) \rightarrow )E)$</td>
<td>0.5</td>
</tr>
<tr>
<td>$E( \rightarrow ( $</td>
<td>0.0</td>
</tr>
<tr>
<td>$E( \rightarrow dE( $</td>
<td>0.5</td>
</tr>
<tr>
<td>$E( \rightarrow xE( $</td>
<td>0.5</td>
</tr>
<tr>
<td>$E( \rightarrow (E( $</td>
<td>0.5</td>
</tr>
</tbody>
</table>
\[ G = (V_N, V_T, P, SS), \text{ where} \]

\[ V_N = \{XY|X \text{ and } Y \text{ are alphabets}\} \cup \{S_1|1 = 0,1,\ldots,9\}\]

\[ V_T = \{a, b, c, d, +, x, *, (, )\}\]

\[ P: \]

1. \[ SS + S_0 \]
2. \[ S_0 + BA + KA * \]
3. \[ KA + (BA + BC) \]
4. \[ BA + b BA \]
5. \[ BA + b \]
6. \[ BC + d BC \]
7. \[ BC + d EA \]
8. \[ EA + b EB \]
9. \[ EB + d EB \]
10. \[ EB + d \]
11. \[ SS + S_1 \]
12. \[ S_1 + BD + KB x BG \]
13. \[ KB + BE x BD \]
14. \[ BD + a BD \]
15. \[ BD + a \]
16. \[ BE + c EB \]
17. \[ BG + c BA \]
18. \[ SS + S_2 \]
19. \[ S_2 + KC * \]
20. \[ KC + (BA + BI) \]
21. \[ BI + d BI \]
22. \[ BI + d EE \]
23. \[ EE + c EF \]
24. \[ EF + b BD \]
25. \[ SS + S_3 \]

26. \[ S_3 + KD \]
27. \[ KD + BG x BK \]
28. \[ BK + d EI \]
29. \[ EI + a BA \]
30. \[ SS + S_4 \]
31. \[ S_4 + BA + KF \]
32. \[ KE + (BA + EB) \]
33. \[ KF + KE x EB \]
34. \[ SS + S_5 \]
35. \[ S_5 + BA + KG \]
36. \[ KG + BQ x BA \]
37. \[ BQ + c BQ \]
38. \[ BQ + c \]
39. \[ S_2 + KH * \]
40. \[ KH + (BA + EB x BI) \]
41. \[ EF + b EF \]
42. \[ SS + S_6 \]
43. \[ S_6 + EF + KI x BQ \]
44. \[ KI + BA x BD \]
45. \[ S_1 + BD + KJ x BA \]
46. \[ KJ + BQ x BD \]
47. \[ SS + S_7 \]
48. \[ S_7 + BA + KK + KL \]
49. \[ KK + BA x EB \]
50. \[ KL + BA x BA \]

*Table 4.6 The grammar G*
Table 4.6 Continued

<table>
<thead>
<tr>
<th></th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>$S_2 \rightarrow KM$</td>
</tr>
<tr>
<td>52</td>
<td>$KM \rightarrow (BA + EB \times BR)$</td>
</tr>
<tr>
<td>53</td>
<td>$BR \rightarrow d BR$</td>
</tr>
<tr>
<td>54</td>
<td>$BR \rightarrow d EG$</td>
</tr>
<tr>
<td>55</td>
<td>$EG \rightarrow b EG$</td>
</tr>
<tr>
<td>56</td>
<td>$EG \rightarrow b EH$</td>
</tr>
<tr>
<td>57</td>
<td>$EH \rightarrow a EB$</td>
</tr>
<tr>
<td>58</td>
<td>$SS \rightarrow S_B$</td>
</tr>
<tr>
<td>59</td>
<td>$S_B \rightarrow KN$</td>
</tr>
<tr>
<td>60</td>
<td>$KN \rightarrow BG \times BS$</td>
</tr>
<tr>
<td>61</td>
<td>$BS \rightarrow a EF$</td>
</tr>
<tr>
<td>62</td>
<td>$S_6 \rightarrow BA + KL \times BQ$</td>
</tr>
<tr>
<td>63</td>
<td>$S_5 \rightarrow BA + KJ$</td>
</tr>
<tr>
<td>64</td>
<td>$S_3 \rightarrow KO + KL$</td>
</tr>
<tr>
<td>65</td>
<td>$KO \rightarrow BG \times BT$</td>
</tr>
<tr>
<td>66</td>
<td>$BT \rightarrow d BD$</td>
</tr>
<tr>
<td>67</td>
<td>$S_6 \rightarrow BA + KL \times BU$</td>
</tr>
<tr>
<td>68</td>
<td>$BU \rightarrow b BQ$</td>
</tr>
<tr>
<td>69</td>
<td>$S_0 \rightarrow BA + KP$</td>
</tr>
<tr>
<td>70</td>
<td>$KP \rightarrow (BA + BV)$</td>
</tr>
<tr>
<td>71</td>
<td>$BV \rightarrow d BV$</td>
</tr>
<tr>
<td>72</td>
<td>$BV \rightarrow d EF$</td>
</tr>
<tr>
<td>73</td>
<td>$S_7 \rightarrow BA + KK + KK \times BA$</td>
</tr>
<tr>
<td>74</td>
<td>$S_1 \rightarrow BD + KJ \times BQ$</td>
</tr>
<tr>
<td>75</td>
<td>$S_0 \rightarrow BA + KR$</td>
</tr>
<tr>
<td>76</td>
<td>$KR \rightarrow (CA)$</td>
</tr>
<tr>
<td>77</td>
<td>$CA \rightarrow b CA$</td>
</tr>
<tr>
<td>78</td>
<td>$CA \rightarrow b EI$</td>
</tr>
<tr>
<td>79</td>
<td>$EI \rightarrow a EJ$</td>
</tr>
<tr>
<td>80</td>
<td>$EJ \rightarrow d EK$</td>
</tr>
<tr>
<td>81</td>
<td>$EK \rightarrow c EG$</td>
</tr>
<tr>
<td>82</td>
<td>$S_3 \rightarrow KS$</td>
</tr>
<tr>
<td>83</td>
<td>$KS \rightarrow BA \times BK$</td>
</tr>
<tr>
<td>84</td>
<td>$BK \rightarrow d BK$</td>
</tr>
<tr>
<td>85</td>
<td>$S_5 \rightarrow BA + KT$</td>
</tr>
<tr>
<td>86</td>
<td>$KT \rightarrow BU \times BD$</td>
</tr>
<tr>
<td>87</td>
<td>$S_0 \rightarrow BA + KU$</td>
</tr>
<tr>
<td>88</td>
<td>$KU \rightarrow (BA + EJ)$</td>
</tr>
<tr>
<td>89</td>
<td>$EJ \rightarrow d EJ$</td>
</tr>
<tr>
<td>90</td>
<td>$S_8 \rightarrow KW$</td>
</tr>
<tr>
<td>91</td>
<td>$KW \rightarrow BA \times BS$</td>
</tr>
<tr>
<td>92</td>
<td>$S_4 \rightarrow BA + KZ$</td>
</tr>
<tr>
<td>93</td>
<td>$KY \rightarrow (EB \times KE)$</td>
</tr>
<tr>
<td>94</td>
<td>$KZ \rightarrow KY \times EB$</td>
</tr>
<tr>
<td>95</td>
<td>$S_5 \rightarrow BA + MA$</td>
</tr>
<tr>
<td>96</td>
<td>$MA \rightarrow BG \times BD$</td>
</tr>
<tr>
<td>97</td>
<td>$S_7 \rightarrow BA + KK + MC$</td>
</tr>
<tr>
<td>98</td>
<td>$MC \rightarrow BD \times BA$</td>
</tr>
<tr>
<td>99</td>
<td>$S_7 \rightarrow EF + MD + MC$</td>
</tr>
<tr>
<td>100</td>
<td>$MD \rightarrow EI \times EB$</td>
</tr>
<tr>
<td>101</td>
<td>$S_1 \rightarrow BD + MA \times BQ$</td>
</tr>
<tr>
<td>102</td>
<td>$S_0 \rightarrow BA + ME$</td>
</tr>
<tr>
<td>103</td>
<td>$ME \rightarrow (BA + EB \times CB)$</td>
</tr>
<tr>
<td>104</td>
<td>$CB \rightarrow d CB$</td>
</tr>
<tr>
<td>105</td>
<td>$CB \rightarrow d EL$</td>
</tr>
<tr>
<td>106</td>
<td>$EL \rightarrow c EH$</td>
</tr>
<tr>
<td>107</td>
<td>$EH \rightarrow a EH$</td>
</tr>
<tr>
<td>108</td>
<td>$S_6 \rightarrow BA + KI + MF$</td>
</tr>
<tr>
<td>109</td>
<td>$MF \rightarrow BD \times BG$</td>
</tr>
<tr>
<td>110</td>
<td>$EI \rightarrow a EI$</td>
</tr>
<tr>
<td>111</td>
<td>$S_5 \rightarrow EF + KT$</td>
</tr>
<tr>
<td>112</td>
<td>$S_1 \rightarrow BD + KJ \times BQ$</td>
</tr>
</tbody>
</table>
Table 4.6 Continued

113  \( S_2 \rightarrow KU \)
114  \( S_5 \rightarrow BK + MG \)
115  \( MG \rightarrow BG \times EF \)
116  \( S_4 \rightarrow BA + MH \)
117  \( MH \rightarrow KE \times EB \)
118  \( S_5 \rightarrow BA + MI \)
119  \( MI \rightarrow BQ \times BD \)
120  \( S_7 \rightarrow BA + MJ \times EB + MK \)
121  \( MJ \rightarrow EB \times EI \)
122  \( MK \rightarrow BA \times BA \)
123  \( S_2 \rightarrow ML \)
124  \( ML \rightarrow ( BA + CC ) \)
125  \( CC \rightarrow d CC \)
126  \( CC \rightarrow d EN \)
127  \( EN \rightarrow c EA \)
128  \( SS \rightarrow S_9 \)
129  \( S_9 \rightarrow MM \)
130  \( MM \rightarrow BA \times CD \)
131  \( CD \rightarrow d CD \)
132  \( CD \rightarrow d EQ \)
133  \( EQ \rightarrow a EQ \)
134  \( EQ \rightarrow a ER \)
135  \( ER \rightarrow b BE \)
136  \( S_3 \rightarrow MN + KL \)
137  \( MN \rightarrow BA \times BT \)
138  \( S_1 \rightarrow EF + KG \times BQ \)
139  \( S_7 \rightarrow EF + KI + MP \)
140  \( MP \rightarrow EI \times BA \)
4.6. Conclusions and Remarks

In Sections 4.4 and 4.5, we have demonstrated that, by using an error-correcting parser, the distance between a syntactic pattern and a group of syntactic patterns can be determined. Such a distance can be used for the nearest neighbor recognition and the cluster analysis for syntactic patterns. Essential parts in both applications include grammatical inference and error-correcting parsing. If the correct classifications of pattern samples are known, the proposed nearest neighbor syntactic recognition rule can be applied to determine the classification and structural description of an unknown pattern. Using Algorithm 2.3 to determine the average distance between a sentence and the K-nearest sentences in the language, the proposed rule for a single nearest neighbor can be easily extended to the K-nearest neighbors.

When the correct classifications of pattern samples are unknown, a non-supervised procedure must be used. In this case, the proposed clustering procedure can be applied. Using error-correcting parsers in cluster analysis, after the clustering result is obtained, we could only implement conventional non-error-correcting parsers for recognition. Furthermore, the grammars inferred could be in finite-state form, the construction of conventional parsers for finite-state grammars is straightforward, and the parsing procedure is, in general, deterministic and efficient. The proposed clustering procedure can certainly be extended to syntactic patterns represented by trees since tree grammar inference procedures [93] and error-correcting tree automata have already been developed. When a general error-correcting parser is used, the computer time required for clustering analysis could be slow. (For
example, using a CDC 6500 computer with FORTRAN IV programming language, the average computer time for analyzing each pattern in Experiment (iii) is 37 sec.) Nevertheless, this requirement may not be critical since a clustering algorithm is used primarily for pattern analysis rather than recognition. Besides, inference procedures for special grammars [74], (such as finite-state and precedence grammars) can always be employed to speed up the analysis.

The grammar inferred for each cluster often generates not only the sentences (syntactic patterns) already in the cluster, but also sentences with similar structures. For example, the grammar $G_1^{(0)}$ in Table 4.3 is an inferred grammar for Pattern 1, $b b b + (b + d d b d d)^*$. However, $L(G_1^{(0)}) = \{b^{n_1} + (b^{n_2} + d^{n_3} b d^{n_4})^*|n_1, n_2, n_3, n_4 \text{ are positive integers}\}$ which represents character $P$ of different sizes (Cluster 0). Consequently, the unweighted distance between Pattern 18 (a character $P$) and Cluster 0 is 2 although the unweighted distance between Pattern 18 and Pattern 1 is 4. For this reason, the clustering procedure proposed in Section 4.5 appears to be more effective and flexible than that of computing the distance between an input pattern sample and a set of prototype or reference patterns on a sentence-by-sentence basis.

With syntactic or linguistic representations of data being more and more common in pattern recognition, speech and language analysis, and database systems [98-101], nearest neighbor recognition rule, and clustering procedures for syntactic patterns should find their applications in these areas.
CHAPTER 5
A SYNTACTIC APPROACH TO TEXTURE ANALYSIS

5.1 Introduction

Research on texture analysis—modeling, synthesis, classification, and discrimination—has received increasing attention in recent years [78]. Texture is a term for the quality of a surface. The feature that dominates a texture scene is the repetitive or quasi-repetitive pattern. Texture information is valuable in scene segmentation, especially in those cases in which the contrast between the object to be observed and the background is poor. A survey of research in this area can be found in Zucker [79]. Applications of texture analysis include terrain classification [80-81], radiographic image interpretation [82,83], microscopic cell image analysis [84-86], materials inspection [87], and many others.

Most of the previous research has concentrated on the statistical approach [80-90]. In this approach, statistical properties are calculated from a set of local measurements taken from the pattern. Weszka, Dyer, and Rosenfeld [81] give a comparative study of several frequently used features for texture classification.

An alternative approach to the statistical one for texture analysis is the structural approach [78]. A texture is considered to be defined by subpatterns which occur repeatedly according to a set of well-defined placement rules within the overall pattern. Furthermore, the subpattern itself is made of structured elements. Compared with the statistical
approach, the structural approach appears to be easier to interpret.

Zucker [79] proposed the idea of texture modeling in terms of an ideal texture and its transformations. According to Zucker, an ideal texture is a deep, unobservable, highly-structured perfect pattern in which local primitives (fundamental building blocks) are extended into a global structure such as regular tesselation. He believes that transformation rules can be defined from a representation of an ideal texture to that of a natural texture. In our work, we shall propose a tree grammar which defines such rules.

Carlucci [91] has formulated a system called "texture language" for the description of some simple repetitive subpatterns such as polygons or open polygonals. Texture patterns are treated as graphs with the basic elements in the texture language representation being lines and vertices. The structure of a subpattern is then represented as a tree. From a practical point of view, Carlucci's system may encounter difficulties during preprocessing, such as difficulty in the extraction of lines and vertices in a texture region.

Ehrich and Foith [42], propose a tree language approach for the description of the structure of waveforms called "relational trees." They believe that information about textures in an image can be obtained by sequential analysis of individual scan lines. The change of gray levels of a scan line gives a random waveform which can be represented by a relational tree.

We shall propose a texture model based on the structural approach. In this model, a texture pattern is divided into fixed-size windows. Repetition of subpatterns or a portion of a subpattern may appear in a
window. For all cases, we shall treat a windowed pattern as a subpattern. A tree grammar is then used to characterize windowed patterns of the same class. This model can be used for texture synthesis as well as discrimination. Since the windowed patterns are also a part of the global structure of the texture, a higher level of syntax rules can also be constructed for the arrangement of windowed patterns.

5.2 A Syntactic Model for Texture Analysis

We propose the following syntactic model for texture analysis: its primitive, window, tree representation, and tree grammar being described here.

5.2.1 The Primitive

We choose a single pixel with different gray levels to be the pattern primitive. For a picture of $L$ gray levels, we have $L$ different primitives. However, the size of a primitive can certainly be larger than a single pixel. For example, a window of $n \times n$ pixels with a relatively uniform gray level would be a good primitive.

5.2.2 The Window

From the structural point of view, texture is the placement of structured subpatterns. (See Figure 5.1(b)). However, in a natural scene, the exact boundary of a subpattern is usually vague and unidentifiable. Often, subpatterns of the same texture appear in various sizes, shapes, or brightness. In some cases, there even exists a situation in which there are no well-defined subpatterns. For examples of this see Figures 5.1 (c) and 5.1 (d). In addition, the placement of subpatterns can be irregular and distorted such as shown in Figure 5.1 (a). Nevertheless,
Figure 5.1(a) Pattern D22. Reptile Skin.

Figure 5.1 Four texture patterns obtained from digitizing pictures found in Brodatz's book, *Textures.*
Figure 5.1(b) Pattern D34. Netting.
Figure 5.1(c) Pattern D38. Water.
Figure 5.1(d) Pattern D60. Woodgrain.
Figure 5.2 Two tree structures for texture modeling.
Figure 5.3 Pattern and its tree representation.
a small subframe of the overall texture pattern does maintain some of the characteristics of the texture. To make the syntactic approach practically feasible, pictures are divided into fixed-size windows. A grammar is then used to characterize the windowed pattern of the given texture. Assuming that the window is of size \( k \times k \), there are \( k^2 \) possible patterns. The set of all the windowed patterns of a particular texture is a subset of the \( k^2 \) patterns. Consequently, a high-dimensional regular grammar, for example, a tree grammar, is suitable for the characterization of texture patterns.

5.2.3 The Tree Representation

Before we construct the tree grammar for a texture pattern, each windowed pattern is first transformed into a tree representation. Each pixel in a \( k \times k \) window corresponds to a node on its tree representation. Hence, a pattern primitive becomes the assigned label to its corresponding node. For implementation, a tree structure can be arbitrarily chosen, but is then fixed for all windows during the process. That is, for all tree representations, the tree structure is the same, but node labels are different. Two convenient tree structures are suggested in Figures 5.2 (a) and 5.2 (b) where the window size is 9 x 9. We shall refer to them as Structure A and Structure B, respectively. Clearly, a different choice of tree structure will result in a different tree representation for the same window. This choice will influence the complexity as well as the effectiveness of the constructed tree grammar.

Example 5.1. The pattern shown in Figure 5.3 (a) has the tree representation shown in Figure 5.3 (b) if Structure A is used.
5.2.4 The Tree Grammar

Example 5.2. The following tree grammar $G_1$ will generate the tree representation shown in Figure 5.3 (b):

$G_1 = (V_1,r,P_1,A_1)$ over $\langle \Sigma, r \rangle$

$V_1 = \{A_1, A_2, A_3, A_4, A_5, A_6, N_0, V_0, 1, 0\}$

$\Sigma = \{ 0, 1 \}$

$r = \{0, 1, 2, 3\}$

$P_1$:

- $A_1 \rightarrow 1$
  - $N_0 \quad A_2 \quad N_0$

- $A_2 \rightarrow 1$
  - $N_0 \quad A_3 \quad N_0$

- $A_3 \rightarrow 1$
  - $N_0 \quad A_4 \quad N_0$

- $A_4 \rightarrow 1$
  - $N_0 \quad A_5 \quad N_0$

- $A_5 \rightarrow 1$
  - $V_0 \quad A_6 \quad V_0$

- $A_6 \rightarrow 1$
  - $N_0 \quad A_6 \quad N_0$
Example 5.3. Grammar $G_1$ in Example 5.2 will only accept patterns of a cross in the middle of the $9 \times 9$ window such as the texture pattern shown in Figure 5.4. In a natural texture, we will most likely have some distortions of the perfect pattern such as the pattern shown in Figure 5.5. Grammar $G_2$ will generate patterns having a shifting of the cross in Figure 5.3 (a) anywhere within the window.

$$G_2 = (V_2, r, P_2, S_2) \text{ over } <\Sigma, r>$$

$$V_2 = \{A_1, A_2, C_1, C_2, D_1, D_2, E_1, E_2, F_1, F_2, G_1, G_2, H_1, H_2, I_1, I_2, J_1, J_2, N_0, N_1, N_2, N_3, N_4, V_0, I, O\}$$

$$S_2 = \{A_1, C_1, D_1, E_1, F_1, G_1, H_1, I_1, J_1\}$$

$P_2$:

$$A_1 \rightarrow \begin{array}{c|c|c|c} 1 & 1 & 1 \\ \hline N_0 & A_1 & N_0 & V_0 & A_2 & V_0 & V_0 & V_0 \\ \end{array}$$

$$A_2 \rightarrow \begin{array}{c|c|c|c} 1 & 1 & 1 \\ \hline N_0 & A_2 & N_0 & N_0 & N_0 \\ \end{array}$$

$$C_1 \rightarrow \begin{array}{c|c|c|c} 0 & 1 & 1 & 1 \\ \hline N_1 & C_1 & N_0 & V_0 & C_2 & V_0 & V_0 & V_0 \\ \end{array}$$
Figure 5.4 A regular texture pattern.

Figure 5.5 A distorted texture pattern which can be generated by $G_2$. 
\[ \begin{align*}
H_1' & \rightarrow 0 \quad ; \quad 1 \quad ; \quad 1 \\
   & \quad \quad N_0 \quad H_1 \quad N_2 \quad \quad V_0 \quad H_2 \quad V_0 \quad \quad V_0 \quad V_0 \\
H_2' & \rightarrow 0 \quad ; \quad 0 \\
   & \quad \quad N_0 \quad H_2 \quad N_2 \quad \quad N_0 \quad N_2 \\
l_1' & \rightarrow 0 \quad ; \quad 1 \quad ; \quad 1 \\
   & \quad \quad N_0 \quad l_1 \quad N_3 \quad \quad V_0 \quad l_2 \quad V_0 \quad \quad V_0 \quad V_0 \\
l_2' & \rightarrow 0 \quad ; \quad 0 \\
   & \quad \quad N_0 \quad l_2 \quad N_3 \quad \quad N_0 \quad N_3 \\
J_1' & \rightarrow 0 \quad ; \quad 1 \quad ; \quad 1 \\
   & \quad \quad N_0 \quad J_1 \quad N_4 \quad \quad V_0 \quad J_2 \quad V_0 \quad \quad V_0 \quad V_0 \\
J_2' & \rightarrow 0 \quad ; \quad 0 \\
   & \quad \quad N_0 \quad J_2 \quad N_4 \quad \quad N_0 \quad N_4 \\
N_0' & \rightarrow 0 \quad ; \quad 0 \\
   & \quad \quad N_0 \\
N_1' & \rightarrow 1 \quad ; \quad 1 \\
   & \quad \quad N_0 \\
N_2' & \rightarrow 0 \quad ; \quad 0 \\
   & \quad \quad N_1 \\
N_3' & \rightarrow 0 \quad ; \quad N_2
\end{align*} \]
The distorted pattern shown in Figure 5.5 can be accepted by $G_2$.

5.3 Illustrative Examples of Texture Synthesis

In Section 5.2, a syntactic model is presented for describing windowed patterns. The global structure of the overall texture pattern depends on the arrangement of windowed patterns. In Example 5.3, we constructed the grammar $G_2$ for the acceptance of the texture pattern shown in Figure 5.5. However, when $G_2$ is used for generation, numerous patterns might be produced, one of which is shown in Figure 5.6. Therefore, in order to preserve the coherence between windows, a set of higher level syntax rules is necessary in which the windowed pattern is treated as a primitive, and the overall texture can be represented as a tree which decides the placement of windowed patterns.

In this section, we will illustrate the synthesis of patterns D22, D34, D38, and D68 from Brodatz's Textures [92]. Figures 5.1 (a), (b), (c), and (d) are digitized pictures with resolutions of 400µ, 400µ, 100µ and 400µ, respectively, of the above four patterns. For simplicity, we use only two primitives: black as primitive "1," and white as primitive "0." By setting a threshold for gray levels in Figure 5.1, we obtain four binary pictures, Figures 5.7 (a), (b), (c), and (d) for patterns D22, D34, D38, and D68, respectively.
Figure 5.6 A random texture pattern which could be generated by $G_2$. 
Figure 5.7(a) Binary picture of pattern D22.

Figure 5.7 Binary pictures of patterns in Figure 5.1.
Figure 5.7(b)  Binary picture of pattern D3̄.
Figure 5.7(c) Binary picture of pattern D38.
Figure 5.7(d) Binary picture of pattern D68.
5.3.1 Regular Tesselation

The texture pattern D3/4 is a hexagonally tesselated pattern which is nearly what Zucker called "ideal texture."

Example 5.4. A syntheses of hexagonal tesselation is as follows: Assume that we have two windowed patterns; namely, A1 and C1 shown in Figures 5.8 (a) and (b), respectively, with window-size 9 x 9. Tree grammar G3 generates the tree representations, A1 and C1, using Structure A described in Section 5.2.3. Tree grammar G3′ generates the placement rule for A1 and C1. The placement rule is given in Structure B. G3 and G3′ are given as follows:

\[
G_3 = (V_3, r, P_3, \{A_1, C_1\}) \text{ over } \Sigma, r
\]

\[
V_3 = \{A_1, A_2, A_3, A_4, A_5, A_6, A_7, C_1, C_2, C_3, C_4, C_5, C_6, C_7, N_1, N_2, N_3, N_4, N_0, 1, 0\}
\]

\[
P_3:
\]

\[
A_1 + \quad 1
\]

\[
A_2 + \quad 1
\]

\[
A_3 + \quad 1
\]

\[
A_4 + \quad 1
\]
Figure 5.8 Windowed patterns of hexagonal tesselation.

A₁   C₁   A₁   C₁
(a)

C₁
(b)

Figure 5.9 The generation of a hexagonal pattern by using grammars \( G_3 \) and \( G_3' \).
$G_3' = (V_3', r', P_3', X)$ over $<\Sigma', r'>$

$V_3' = \{X, Y, Z, A_1, C_1\}$

$\Sigma' = \{A_1, C_1\}$, $r' = \{0, 1, 2\}$

$P_3'$:

\[
\begin{align*}
X & \to \begin{cases} 
A_1 & \text{with probability } \frac{1}{3} \\
X & \text{with probability } \frac{2}{3}
\end{cases} \\
Y & \to \begin{cases} 
C_1 & \text{with probability } \frac{1}{3} \\
Z & \text{with probability } \frac{2}{3}
\end{cases}
\end{align*}
\]
The generating procedure using the two-level syntax rules, $G_3$ and $G_3'$ is illustrated in Figure 5.9. The generated pattern is shown in Figure 5.10.

Example 5.5. In Example 5.4, we assumed that the window size matched the size of the hexagonal subpattern. However, the 9 x 9 window in Example 5.4 does not perfectly match the pattern D34 in Figure 5.7 (b). An improvement is shown in Figure 5.11 (b), where the hexagons have a similar size to that of pattern D34. In Figure 5.11 (b), window frames have been drawn in so that the windowed patterns and their repetitive order can be shown clearly. There are 20 different windowed patterns within each heavily lined area in a 4 x 5 arrangement. The larger pattern, made up of the 20 windows, also repeats itself.

The grammar $G_4$ that generates the 20 windowed patterns, is given on the left-hand side of Appendix E-1. Figure 5.11 (a) shows the placement rule in Structure B for the pattern in Figure 5.11 (b). The symbol in each cell of Figure 5.11 (a) belongs to the set of starting symbols in grammar $G_4$. From each starting symbol, the corresponding windowed pattern of Figure 5.11 (b) can be generated. The grammar that generates the placement rule is also given in Appendix E-1 as grammar $G_4'$.

Example 5.6. The uneven brightness in pattern D34, e.g., darker for horizontal lines and lighter for diagonal lines, can be simulated by using a stochastic grammar. Figure 5.12 is the resulting
Figure 5.10 The generated regular hexagonals.
Figure 5.11 Generated regular hexagonals.
Figure 5.12  Simulated result of pattern D34.
pattern from using stochastic grammar $G_{S4}$ in the generation of the "noisy version." The grammar $G_{S4}$ is given on the right-hand side in Appendix E-1.

In this subsection, three examples were given: (1) to illustrate the synthesis of an ideal texture for a matching sized window and sub-pattern, (2) for an unmatched size window and subpattern, and (3) for a noisy version. However, the noisy version described in Example 5.6 is the result of local noise. In the following subsection, we shall describe the synthesis of a global structure-distorted texture pattern.

5.3.2 Irregular Tessellation

Let us examine pattern D22 in Figure 5.7 (a). We may consider that pattern D22 is the result of twisting the regular tesselation of an ideal texture such as the pattern shown in Figure 5.13. From a single window the trend of distortion cannot be fully detected. For texture synthesis, such a global distortion can be treated as a problem of the placement of windowed patterns.

Example 5.7. The regular tesselated pattern shown in Figure 5.13 is composed of two basic patterns shown in Figure 5.14 (a) and (b). A distorted tesselation can result from shifting a series of basic patterns in one direction. Let us use the set of patterns resulting from shifting a basic pattern as the set of primitives. There will be 81 such windowed pattern primitives. We shall refer to them simply as primitives in this example. Figure 5.15 shows several of them. Each primitive is given a name of two symbols. "X_{i}," where $X \in \{A,B,C,D,E,F,G,H,I\}$, $i \in \{1,2,\ldots,9\}$. Starting from $X_{i}$,
Figure 5.13 The ideal texture of pattern D34.
Figure 5.14 Basic pattern of Figure 5.13.

Figure 5.15 Windowed pattern primitives.
the pattern resulting from shifting one column to the left will be
named \( X_{i+1} \), and the pattern resulting from shifting one row up will
be named \( Y_i \). Grammar \( G_5 \) in Appendix E-2 is constructed for the
generation of the 81 primitives.

Several synthesis results are given in Figure 5.16 (a), (b), and
(c). Tree representations using Structure B that decide the placement
of windowed patterns are shown at the left-hand side of each pattern.

Using the same idea as that in Example 5.6, a stochastic grammar can
be used to add local distortions. An example of this is shown in
Figure 5.16 (d). We can also construct a grammar for the placement of
windowed patterns for certain types of structure distortion. For
example, a twisted upward or downward pattern, or an insertion of an
extraneous row of subpatterns, etc.

5.3.3 Random Pattern

The texture pattern \( D_{38} \) and \( D_{68} \) in Figures 5.7 (c) and (d) show a
higher degree of randomness than \( D_{22} \) and \( D_{34} \). No clear tesselation or
subpattern exists in the pattern.

Example 5.8. The water waves in pattern \( D_{38} \) can be described as a
belt extending in the horizontal direction, varying in width and
twisting upward or downward. Assuming that, at most, one belt can
appear in a window, we shall use Structure A for tree representation
and a stochastic tree grammar \( G_6 \) to describe such patterns. In each
production rule in \( G_6 \), the left-hand side nonterminal is the present
state and the right-hand side generates the width and the position
of the belt that the present state represented, as well as the next
state. Figure 5.17 illustrates this generation process. The
Figure 5.16 Synthesis results of pattern D22.

<table>
<thead>
<tr>
<th>A₁</th>
<th>A₂</th>
<th>A₃</th>
<th>A₄</th>
<th>A₅</th>
<th>A₆</th>
<th>A₇</th>
<th>A₈</th>
<th>A₉</th>
<th>A₁₀</th>
<th>A₁₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>B₁</td>
<td>B₂</td>
<td>B₃</td>
<td>B₄</td>
<td>B₅</td>
<td>B₆</td>
<td>B₇</td>
<td>B₈</td>
<td>B₉</td>
<td>B₁₀</td>
<td>B₁₁</td>
</tr>
<tr>
<td>C₁</td>
<td>C₂</td>
<td>C₃</td>
<td>C₄</td>
<td>C₅</td>
<td>C₆</td>
<td>C₇</td>
<td>C₈</td>
<td>C₉</td>
<td>C₁₀</td>
<td>C₁₁</td>
</tr>
<tr>
<td>D₁</td>
<td>D₂</td>
<td>D₃</td>
<td>D₄</td>
<td>D₅</td>
<td>D₆</td>
<td>D₇</td>
<td>D₈</td>
<td>D₉</td>
<td>D₁₀</td>
<td>D₁₁</td>
</tr>
<tr>
<td>E₁</td>
<td>E₂</td>
<td>E₃</td>
<td>E₄</td>
<td>E₅</td>
<td>E₆</td>
<td>E₇</td>
<td>E₈</td>
<td>E₉</td>
<td>E₁₀</td>
<td>E₁₁</td>
</tr>
<tr>
<td>F₁</td>
<td>F₂</td>
<td>F₃</td>
<td>F₄</td>
<td>F₅</td>
<td>F₆</td>
<td>F₇</td>
<td>F₈</td>
<td>F₉</td>
<td>F₁₀</td>
<td>F₁₁</td>
</tr>
<tr>
<td>G₁</td>
<td>G₂</td>
<td>G₃</td>
<td>G₄</td>
<td>G₅</td>
<td>G₆</td>
<td>G₇</td>
<td>G₈</td>
<td>G₉</td>
<td>G₁₀</td>
<td>G₁₁</td>
</tr>
<tr>
<td>H₁</td>
<td>H₂</td>
<td>H₃</td>
<td>H₄</td>
<td>H₅</td>
<td>H₆</td>
<td>H₇</td>
<td>H₈</td>
<td>H₉</td>
<td>H₁₀</td>
<td>H₁₁</td>
</tr>
</tbody>
</table>

Figure 5.16(a)
<table>
<thead>
<tr>
<th>$H_1$</th>
<th>$H_2$</th>
<th>$I_3$</th>
<th>$A_4$</th>
<th>$A_5$</th>
<th>$I_7$</th>
<th>$H_8$</th>
<th>$H_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_6$</td>
<td>$H_2$</td>
<td>$I_3$</td>
<td>$A_5$</td>
<td>$A_6$</td>
<td>$I_5$</td>
<td>$H_8$</td>
<td>$G_5$</td>
</tr>
<tr>
<td>$F_7$</td>
<td>$G_3$</td>
<td>$H_4$</td>
<td>$A_1$</td>
<td>$A_7$</td>
<td>$H_8$</td>
<td>$G_9$</td>
<td>$F_9$</td>
</tr>
<tr>
<td>$F_8$</td>
<td>$G_4$</td>
<td>$H_5$</td>
<td>$A_2$</td>
<td>$A_7$</td>
<td>$H_8$</td>
<td>$G_9$</td>
<td>$F_9$</td>
</tr>
<tr>
<td>$E_9$</td>
<td>$F_3$</td>
<td>$H_5$</td>
<td>$A_7$</td>
<td>$A_9$</td>
<td>$H_1$</td>
<td>$G_9$</td>
<td>$F_9$</td>
</tr>
<tr>
<td>$E_1$</td>
<td>$F_6$</td>
<td>$H_7$</td>
<td>$A_2$</td>
<td>$A_5$</td>
<td>$H_8$</td>
<td>$G_9$</td>
<td>$F_9$</td>
</tr>
<tr>
<td>$C_2$</td>
<td>$E_4$</td>
<td>$G_3$</td>
<td>$A_1$</td>
<td>$A_9$</td>
<td>$G_9$</td>
<td>$E_5$</td>
<td>$B_2$</td>
</tr>
<tr>
<td>$C_3$</td>
<td>$E_3$</td>
<td>$G_4$</td>
<td>$A_3$</td>
<td>$A_5$</td>
<td>$G_9$</td>
<td>$F_9$</td>
<td>$C_9$</td>
</tr>
</tbody>
</table>

Figure 5.16(c)

Figure 5.16(d)
<table>
<thead>
<tr>
<th>present state</th>
<th>generated pattern</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_1$</td>
<td>![Pattern 1]</td>
<td>$E_1, E_2, C_2, A_1$</td>
</tr>
<tr>
<td>$E_1$</td>
<td>![Pattern 2]</td>
<td>$E_1, E_2, C_2, A_1$</td>
</tr>
<tr>
<td>$C_2$</td>
<td>![Pattern 3]</td>
<td>$C_2, C_1, C_3, E_1$</td>
</tr>
<tr>
<td>$C_2$</td>
<td>![Pattern 4]</td>
<td>$C_2, C_1, C_3, E_1$</td>
</tr>
<tr>
<td>$C_3$</td>
<td>![Pattern 5]</td>
<td>$C_3, C_2, E_2$</td>
</tr>
<tr>
<td>$E_2$</td>
<td>![Pattern 6]</td>
<td>$E_2, E_3, E_1, C_3, G_1$</td>
</tr>
<tr>
<td>$G_1$</td>
<td>![Pattern 7]</td>
<td>$G_1, G_2, E_2, A_1$</td>
</tr>
</tbody>
</table>

Figure 5.17 The syntactic generation of water waves.
grammar $G_6$ is given in Appendix F-1. $G_6$ is also the discrimination grammar for pattern D38 which will be discussed in Section 5.4.

The production rules associated with zero probability are unused rules during pattern generation. They are added for pattern discrimination. The probabilities associated with the production rules in $G_6$ are arbitrarily assigned. By varying the assignment of probabilities, patterns with a different degree of brightness and fluctuation can be generated. Some resulting patterns are shown in Figure 5.18.

Example 5.9. The texture pattern of D68, the wood grain pattern, consists of long vertical lines. It is particularly convenient for syntactic description when Structure A is used for tree representation. The subpattern (vertical line) and its repetition can be fully characterized by the stochastic grammar $G_7$. Therefore, there is no need to generate the overall pattern window by window.

The grammar $G_7$ is given as follows:

$$G_7 = (V_7, r, P_7, A_1) \text{ over } \Sigma, r$$

$$V_7 = \{A_1, N_0, N_1, 0, 1\}$$

$$r = \{0, 1, 2, 3\}$$

$$\Sigma = \{0, 1\}$$

$$P_7:$$

$$A_1 \rightarrow \begin{array}{c} 0 \end{array}, 0.5; \begin{array}{c} 0 \end{array}, 0.05;$$

$$\begin{array}{c} N_0 \ \ \ A_1 \ \ \ N_0 \end{array}, 0.09;$$

$$\begin{array}{c} N_1 \ \ \ A_1 \ \ \ N_0 \end{array}$$
Figure 5.18 Synthesis results of pattern D38.
The density of grains (vertical lines) depends on the probabilities associated with production rules. Pictures in Figure 5.19 are generated from $G_7$ using different probability assignments.

5.4 Texture Discrimination

The proposed texture model can also be used for texture discrimination. In Section 5.3, we illustrated how a texture pattern was generated window-by-window. The construction of a grammar in modeling the variation of size, shape, and brightness, as well as noise and distortion was illustrated by examples. We also discussed in Section 5.2 that a pattern in a small subframe (window) maintains some of the characteristics of the overall texture. Under this assumption, we shall restrict the problem of texture discrimination to the recognition of windowed patterns only. Each picture is processed window-by-window.

5.4.1 Data Preparation

The pattern shown in Figure 5.20 consists of patterns D22, D34, D38, and D68. There are 180 x 180 pixels with 128 gray levels. We shall use two primitives (two gray levels) for discrimination. The picture shown in Figure 5.21 is obtained by setting a threshold at gray level 44. Window frames are drawn in Figure 5.21. The window size is 9 x 9.

5.4.2 Discrimination Grammars

The texture modeling grammars described in Section 5.3 are used for discrimination here. Let the grammar for pattern D22, D34, D38, and D68 be $G_{22}$, $G_{34}$, $G_{38}$, and $G_{68}$, respectively. From the viewpoint of discrimination, we would like to modify the grammar so that overlaps between $L(G_{22})$, $L(G_{34})$, $L(G_{38})$, and $L(G_{68})$ (languages generated from
Figure 5.19 Synthesis results of pattern D68.
Figure 5.20 Pictorial data for texture discrimination.
Figure 5.21 Binary picture of Figure 5.20.
G_{22}, G_{34}, G_{38}, and G_{68}, respectively) will be as small as possible. Whereas, each language itself needs to be as general in characterizing each class of texture as possible. Grammar G_{22}, G_{34}, and G_{68} are given in Appendix F-2, F-3, and F-4, respectively. Grammar G_{38} is the non-stochastic version of grammar G_{6} in Appendix F-1.

5.4.3 Error-Correcting Parsing

The non-convexional parser usually fails to recognize a "noisy" pattern. Although we have tried to construct the discrimination grammars to include as large a variety of patterns as possible, the uncertainty existing in a pattern is impossible to be fully characterized and predicted. An error-correcting parser can be used to improve the classification accuracy. In particular, in this application, we shall use the SPECTA (structure-preserved error-correcting tree automata) as the texture discriminator.

5.4.4 Computation Result

The SPECTA measures the distance between the input tree representation and the texture languages, L(G_{22}), L(G_{34}), L(G_{38}), and L(G_{68}) one by one. Then, the input pattern is classified to the texture class which has the minimum distance.

The result of texture discrimination for the picture in Figure 5.21 is given in Figure 5.22. There are 400 windows. Thirty of them are misrecognized. The misrecognition usually results from the unavoidable overlap between two languages or from the reduction of one language to decrease the overlap.
5.5 Remarks

In this chapter, a syntactic approach for texture modeling is presented. The proposed approach appears to be attractive from the practical point of view. The preprocessing involves picture digitization only. The window operation stores a small subframe of the pattern in the main memory. Thus, the process is manageable by a small memory computer.

The most difficult part comes from the construction of an effective grammar. Since no sophisticated preprocessing is used, the linguistic representations are very sensitive to noise. In constructing a grammar, we would like to consider as many variations of the texture pattern as possible. On the other hand, we also need to keep the grammar as simple (as few nonterminals and production rules) as possible to save storage space. Such a compromise often results in a grammar that generates some excessive sentences, but excludes some possible distortions. That is one reason for the necessity of using an error-correcting parser for picture parsing in texture discrimination. The other reason is the uncertainty existing in the picture making the construction of a grammar difficult in order to fit all the possibilities of a texture class.

All the computation examples are programmed in Fortran IV on a PDP-11/45 computer with a 32K core memory. The SPECTA we designed processes all the branches of a tree from the frontiers to the root in parallel, but it should be programmed in series on a general purpose computer. The process can certainly be speeded up by a specially designed processor.

Automatic grammatical inference procedures for tree languages have been recently studied [93]. By combining an inference algorithm with the proposed discrimination procedure, an automation of the entire training and testing process as proposed in Chapter 4 can be implemented.
Color Code

Pattern D34
Pattern D22
Pattern D68
Pattern D38

Figure 5.22. Discrimination result.
CHAPTER 6
SUMMARY OF RESULTS AND SUGGESTIONS FOR FURTHER RESEARCH

6.1. Summary of Results and Conclusions

The problem of modeling, analysis and reconstruction of noisy and/or distorted syntactic patterns is studied. In syntactic pattern recognition, a pattern is described in terms of its subpatterns, primitives and the relations among them. Segmentation errors and primitive extraction errors can be treated as syntax errors and defined in terms of language transformation rules. Three types of error transformations are defined on strings, namely, substitution, insertion and deletion. Consequently, the parser constructed according to the grammar generating the strings and the three types of transformations is called the error-correcting parser. A stochastic deformation model and stochastic error transformation rules are also proposed. In searching for the most likely correction, the formulation of error-correcting parser (ECP) for context-free languages and context-free programmed languages are based on the minimum-distance criterion for non-stochastic model and the maximum-likelihood criterion for stochastic model.

The error-correcting parsing technique for string languages has been extended to tree languages. In formulating error-correcting tree automata (ECTA), five types of error transformations on trees are defined, namely, substitution, split, stretch, branch and deletion. Two types of ECTA are proposed; a SPECTA corrects substitution errors only, and a GECTA corrects all five types of errors.
By way of using language transformations, the distance between two sentences - strings, or trees, can be determined. This idea provides a necessary tool for the clustering analysis for syntactic patterns. The algorithms of constructing minimum spanning tree and clustering centers in statistical pattern recognition are extended to syntactic patterns. A definition of distance between a sentence and a language is proposed. Based on this definition, a new clustering procedure is proposed, where a grammar is inferred to characterize a formed cluster, and then updated when a new pattern is assigned to the cluster. An error-correcting parser, in this case, is employed to measure the distance between an input syntactic pattern and a formed cluster, or a language. Therefore, the procedure yields not only the clustering results but also the syntax rules characterizing each cluster.

Finally, using the error-correcting parsing techniques, a real data example on texture modeling and discrimination are presented. In texture modeling, the idea of window operation is used. Texture patterns are divided into fixed size windows. Windowed patterns belonging to the same class of texture are then characterized by a tree grammar. This tree grammar is used for texture synthesis as well as discrimination. However, in texture synthesis, the coherence between windowed patterns is also essential to the overall pattern. It is proposed to use a higher level syntax, for example, another tree grammar, as a monitor for the placement of windowed patterns. Consequently, structural distortion can be simulated by changing the placement of windowed patterns, whereas, local noise can be simulated by using stochastic grammars for characterizing windowed patterns. In texture discrimination, a set of SPECTA, where each is constructed for one class of texture, is used as discriminator. An input windowed pattern is analyzed
by the SPECTA's then classified according to the nearest neighbor rule.

6.2. Suggestions for Further Research

The proposed similarity measure and error-correcting parsing scheme provide a formal model and a useful tool for recognition under uncertainty, for example, in a clustering problem where correct classification for all samples are unknown, or in a noisy environment, such as the analysis of texture patterns. To improve the recognition results, the following problems need further investigation:

(1) To improve the parsing efficiency. The weakness of error-correcting parsing is the need of long computing time. In the case of error-correcting parsing for context-free languages, sequential method has been proposed to reduce the computing time. However, further improvement is still needed. In the texture discrimination example, patterns are divided into fixed-size windows, and then each window is classified by using SPECTA. Both window operation and SPECTA are parallel operations. Parallel processing techniques are expected to be very useful in improving the computation efficiency.

(2) The inference of updated grammars. The knowledge of classification obtained from using error-correcting parsers is accumulated by way of updating pattern grammars. Different from the existing grammatical inference procedure which often require all the training data be available at the same time, training data are given sequentially in the grammar updating problem. To update a grammar to be simple and effective is a problem for future study.
(3) The inference of a set of weights associated with error transformations. Using weighted distance can improve the clustering results. The problem of finding a set of appropriate weights from training samples requires further study.

(4) The inference of texture grammars. Due to the noise and variation of texture patterns, it is cumbersome to construct texture grammars manually. To be more practical, the proposed syntactic model for texture analysis needs an efficient inference procedure. There are two problems to be studied; the first is the inference of stochastic tree grammar for windowed patterns, and the second is to infer placement rules for a texture pattern. In the second problem, the regularity, or the repetition of the basic texture patterns has to be determined. Other problems such as choosing a suitable window size based on density or coarseness of textures, and finding the relations between window size, the effectiveness of texture grammar and parsing efficiency, etc. are also interesting problems to be investigated.

(5) To apply the proposed syntactic texture model to analyze real world data, such as aerial photographs, X-ray images and LANDSAT data, etc.
BIBLIOGRAPHY


13. Aho, A. V. and T. G. Peterson, "A Minimum Distance Error-Correcting
Parser for Context-Free Languages," SIAM J. Comput., Vol. 4,
December, 1972.


Vol. 6, No. 11, November, 1963.


17. Wirth, N., "A Programming Language for the 360 Computers," J. ACM,
Vol. 15, January 1968.


22. Hornung, J. J., What the Compiler Should Tell the User, Lecture
Notes in Computer Science, Vol. 21, Ed. by G. Goos and J. Harmants,
Springer-Verlag, 1974.


Language: A Practical Approach," Comm. ACM, Vol. 17, No. 1,

25. Morgan, H. L., "Spelling Correction In System Programs," Comm. ACM,


27. Wagner, R. A., "Order-n Correction for Regular Languages," C. ACM,


73. Todd, W. J. and M. F. Baumgardner, "Land Use Classification of Marion County, Indiana, Data," LARS Information Note 101673, LARS, Purdue University, 1973.


APPENDICIES
APPENDIX A

PROOF OF THE CONSISTENCY OF THE MULTIPLE ERROR MODEL

Assume that deformation probabilities on terminal $a \in \Sigma$ is consistent on a single error model; i.e., equation (2.1) in Section 2.4 is satisfied. By summing over all the cases in equation (2.2) we have,

$$
\sum_{a \in \Sigma} q(a|a) = (q_D(a)) + \{ \sum_{b \in \Sigma} [q_S(b|a) + q_I(b|a)q_D(a)] \\
+ \sum_{l=2}^{\infty} \prod_{j=1}^{l-1} \left( \sum_{b \in \Sigma} q_I(b|a) \right) [1 - q_D(a) - q_I(b|a)] \} \\
+ q_I(b|a)q_D(a)) \} . 
$$

(A1)

From equation (2.7), the first three terms of (A1) can be reduced to

$$
[1 - \sum_{b \in \Sigma} q_I(b|a)] + [\sum_{b \in \Sigma} q_I(b|a)q_D(a)] 
$$

(A2)

and the fourth and fifth term of (A1) can be reduced to

$$
\sum_{l=2}^{\infty} \prod_{j=1}^{l-1} \left( \sum_{b \in \Sigma} q_I(b|a) \right) [1 - q_D(a) - \sum_{b \in \Sigma} q_I(b|a)] \\
+ \sum_{b \in \Sigma} q_I(b|a)q_D(a) \} . 
$$

(A3)
Then,

\[ \sum_{a \in \mathcal{E}} q(a|a) = (A2) + (A3) \]

\[ = 1 - q_1(a) (1 - q_D(a)) \]

\[ + \sum_{\ell=2}^{\infty} \sum_{j=1}^{\ell-1} q_1(a) ([1 - q_1(a)] (1 - q_D(a))) \]

where \( q_1(a) = \sum_{b \in \mathcal{E}} q_1(b|a) \)
APPENDIX B

SEQUENTIAL CLASSIFICATION ALGORITHMS

Let \( \hat{e} \) be a parameter, \( 0 \leq \hat{e} \leq 1 \).

Algorithm 3. Decision Algorithm

**Input:** String \( a_1 a_2 \ldots a_n \).

**Output:** \( C_\ell \), \( 1 \leq \ell \leq k \).

**Method:**

**Step 1.** Set \( j = 0 \) and compute \( r = 1 - \max_i P(C_i) \). If \( r < \hat{e} \), stop and assign class \( C_{\ell_0} \), where \( P(C_{\ell_0}) = \max_i P(C_i) \). If \( r \geq \hat{e} \), set \( j = 1 \) and go to Step 2.

**Step 2.** Parse the \( j \)th input symbol by SPA.

**Step 3.** If Step 2 receives parsing failure flag from all pattern grammars, stop and assign class \( C_{\ell_0} \), where \( p(C_{\ell_0}|a_1 a_2 \ldots a_j a) = \max_i p(C_i|a_1 a_2 \ldots a_j a) \), otherwise go to Step 4.

**Step 4.** If \( j = n \), stop and classify to \( C_{\ell_0} \), where \( p(C_{\ell_0}|a_1 a_2 \ldots a_n) = \max_i p(C_i|a_1 a_2 \ldots a_n) \). If \( j \neq n \), compute \( r = 1 - \max_i p(C_i|a_1 a_2 \ldots a_j a) \), go to Step 5.

**Step 5.** If \( r < \hat{e} \), stop and assign \( C_{\ell_0} \) where \( p(C_{\ell_0}|a_1 a_2 \ldots a_j a) = \max_i p(C_i|a_1 a_2 \ldots a_j a) \), otherwise, if \( j = n \) then stop, otherwise, set \( j = j+1 \). Go to Step 2.
Algorithm 4. Sequential Parsing Algorithm

**Input:** A SCFG $G_1 = (N_1, \Sigma_1, P_1, S_1)$ and an input string $a_1a_2...a_j, 1 \leq j \leq n$.

**Output:** $p(a_1a_2...a_j|C_j)$ and $p(a_1a_2...a_j|C_1)$.

**Method:**

**Step 1.** Set $j=0$, add item $[Z \rightarrow \cdot S_1, 0, 1, 1]$ to $I_j$.

**Step 2.**

(a) If $[A \rightarrow \alpha \cdot B\beta, 1, p, r]$ is in $I_j$ and $B \rightarrow \gamma$ is in $P_1$, add item $[B \rightarrow \gamma, j, q, 0]$ to $I_j$.

(b) If $[A \rightarrow \alpha \cdot I, p_1, r]$ is in $I_j$, for all item in $I_j$ of the form $[B \rightarrow \beta \cdot A\gamma, k, p_2, s]$, add item $[B \rightarrow BA \cdot \gamma, k, p_3, 0]$ to $I_j$ where $p_3 = p_1p_2$, unless an item of the form $[B \rightarrow BA \cdot \gamma, k, q, t]$ is already in $I_j$. If this is the case, set $q = q + p_1p_2$.

**Step 3.**

(a) For each item $I_j$ of the form $[A \rightarrow \alpha \cdot B\beta, I, q, r]$, where $I < j$, find in $I_j$ all the items of the form $[C \rightarrow \mu \cdot A\gamma, k, t, s]$. Suppose that there are $m$ such items with values for $s$ equals to $s_1s_2...s_m$, respectively, set $r = q(s_1 + s_2 + ... + s_m)$.

(b) Locate all items in $I_j$ of the from $[A \rightarrow \alpha \cdot B\beta, j, q, r]$. If there are $n$ such items, number these items such that $k$th of them is denoted as $[A_k \rightarrow \alpha_k \cdot B_k\beta_k, j, q_k, r_k]$. For the $k$th item, locate in $I_j$ all items of the form $[C \rightarrow \mu \cdot A_k\gamma, I, t, s]$. If there are $m_k$ such items, we denote the values for $s$ in these items as $s_1s_2...s_{m_k}$, then $r_k = q_k(s_1 + s_2 + ... + s_{m_k}), k = 1, 2...n$. Note that either $s_1$ is determined by Step 3(a) already, or
Step 4. If \( j = 0 \), go to Step 7; otherwise, go to Step 5.

Step 5. Compute \( p(a_1a_2\ldots a_j|G_j) \) as follow, if an item of the form
\[
[Z \rightarrow S_1 \cdot \cdot \cdot 0,p,r]
\]
is in \( 1_j \), set \( p(a_1a_2\ldots a_j|G_j) = p \), otherwise \( p(a_1a_2\ldots a_j|G_j) = 0 \).

Step 6. Applying \( p(a_1a_2\ldots a_j|G_j) \) and \( p(a_1a_2\ldots a_j|G_j) \) obtained from Step 5 and Step 8 to Algorithm 3, If it is decided to continue, go to Step 7; otherwise, stop and classify.

Step 7. Set \( j = j + 1 \). For each item in \( 1_{j-1} \) of the form
\[
[A \rightarrow a_1a_2\ldots a_j|G_j]
\]
add item \([A \rightarrow a_1a_2\ldots a_j|G_j] \) to \( 1_j \), go to Step 8. If no item of the form \([A \rightarrow a_1a_2\ldots a_j|G_j] \) in \( 1_{j-1} \) exists, set parsing failure flag and \( p(a_1a_2\ldots a_j|G_j) = 0 \), \( p(a_1a_2\ldots a_j|G_j) = 0 \), stop.

Step 8. Locate the items which are added to \( 1_j \) by Step 7. Suppose that there are \( n \) such items, number the \( m \)-th of them as
\[
[A \rightarrow a_1a_2\ldots a_j|G_j] \text{.}
\]
Find all items in \( 1_m \) of the form
\[
[B \rightarrow a_1a_2\ldots a_j|G_j] \text{.}
\]
Suppose that there are \( s \) such items with parameters \( s \) denoting as \( s_1,s_2\ldots s_k \), then
\[
p(a_1a_2\ldots a_j|G_j) + p(a_1a_2\ldots a_j|G_j) = \sum_{m=1}^{k} p_m \sum_{l=1}^{r_m} s_l \text{, go to Step 2.}
\]
APPENDIX C
HIGHWAY GRAMMAR

\[ G_H = (V,P,r,S) \text{ over } \langle E,r \rangle \text{ where,} \]

- \( V = \{S,H_0,X_0,A_1 \ldots 8, B_1 \ldots 8, C_1 \ldots 3, D_1 \ldots 3, E_1 \ldots 7, F_1 \ldots 7, H_1 \ldots 7, M_1 \ldots 7, h,b,\$\} \)
- \( r(\$) = \{1\} \)
- \( r(h) = \{0,1,3\} \)
- \( r(b) = \{0,1,3\} \)

\[ P: \]

\[
S \rightarrow \begin{align*}
&\$ \quad , \quad \$ \quad , \quad 1 = 1 \ldots 8 \\
&\quad | \quad A_1 \quad B_1 \\
&\quad | \quad C_1 \quad D_1 \\
&\quad | \quad E_1 \quad F_1 \\
&\quad | \quad M_1 \\
&\quad | \quad H_0 \quad X_0 
\end{align*}
\]
Figure C-1  Typical patterns generated from $G_H$
Figure C-1 Continued

(f) Group F, M
APPENDIX D
THE GRAMMARS AND RESULTS OF THE
CHARACTER RECOGNITION EXAMPLE

D.1 Character Grammars

(a) Character A

\[ G_A = (V_A, r_A, P_A, A) \] where

\[ V_A = \{A, A_1, A_2, N_a, N_d, N_c\}, \quad \Sigma_A = \{a, c, d\} \]

\[ r_A(1) = \{2\}, \quad r_A(a) = \{0, 2\}, \quad r_A(c) = \{0, 1\}, \quad r_A(d) = \{0\} \]

\[ P_A: \]

\[ A \rightarrow \]

\[ A_1 \]

\[ A_2 \]

\[ A_1 \rightarrow \]

\[ a \]

\[ N_a \]

\[ N_d \]

\[ A_2 \rightarrow \]

\[ c \]

\[ N_c \]

\[ N_a \rightarrow a, \quad N_d \rightarrow d, \quad N_c \rightarrow c \]

(b) Character C

\[ G_C = (V_C, r_C, P_C, C) \] where

\[ V_C = \{C, C_1, C_2, C_3, N_d, N_f\}, \quad \Sigma_C = \{a, b, d, f\} \]

\[ r_C(1) = \{2\}, \quad r_C(a) = \{1\}, \quad r_C(b) = \{1\}, \quad r_C(d) = \{0, 1\}, \quad r_C(f) = \{0\} \]
\( P_C: \)

\[
\begin{array}{c}
\text{C} \\
\downarrow & \\
\text{C}_1 & \text{N}_d \\
\end{array}
\]

\( C_1 + \)

\[
\begin{array}{c}
\text{a} \\
\text{C}_2 \\
\end{array}
\]

\( C_2 + \)

\[
\begin{array}{c}
b \\
\text{C}_3 \\
\end{array}
\]

\( C_3 + \)

\[
\begin{array}{c}
d \\
\text{N}_f \\
\end{array}
\]

\( N_d \rightarrow d, \quad N_f \rightarrow f \)

(c) Character D

\( G_D = (V_D, r_D, P_D, D) \)

\( V_D = \{D, D_1, D_2, D_3, D_4, N_b, N_d\} \), \( \Sigma_D = \{1, b, d\} \)

\( r_D(1) = \{2\}, \quad r_D(b) = \{0, 1\}, \quad r_D(d) = \{0, 1\} \)

\( P_D: \)

\[
\begin{array}{c}
\text{D} \\
\downarrow & \\
\text{D}_1 & \text{D}_2 \\
\end{array}
\]

\( D_1 + \)

\[
\begin{array}{c}
b \\
\text{D}_3 \\
\end{array}
\]

\( D_3 + \)

\[
\begin{array}{c}
b \\
\text{N}_d \\
\end{array}
\]

\( \text{N}_d \rightarrow d \)
(d) Character $E$

$G_E = (V_E, r_E, P_E, E)$

$V_E = \{E, E_1, E_2, E_3, N_d, E\}$, $r_E = \{1, b, d\}$

$r_E(1) = \{2, 3\}$, $r_E(b) = \{1, 2\}$, $r_E(d) = \{0, 1\}$

$P_E$: $E \rightarrow E_1 \quad \quad E_3$

$E_1 \rightarrow E_2 \quad \quad E_3$

$E_2 \rightarrow E_3$

$E_3 \rightarrow N_d$

$N_d \rightarrow d$, $N_b \rightarrow b$
(e) Character H

\[ \chi_H = (V_H, r_H, P_H) \]

\[ V_H = \{H, H_1, H_2, N_b, N_f\}, \quad \Sigma_H = \{1, b, d, f\} \]

\[ r_H(1) = \{2\}, \quad r_H(b) = \{0, 2\}, \quad r_H(d) = \{2\}, \quad r_H(f) = \{0\} \]

\[ P_H: \quad H \rightarrow \begin{array}{c}
1 \\
H_1
\end{array} \]

\[ H_1 \rightarrow \begin{array}{c}
\top \\
N_b & H_2
\end{array} \]

\[ H_2 \rightarrow \begin{array}{c}
\top \\
N_b & N_f
\end{array} \]

\[ N_b \rightarrow b \]

\[ N_f \rightarrow f \]
Appendix D.2. Classification Results of 26 Test Patterns

<table>
<thead>
<tr>
<th>INPUT CHARACTER</th>
<th>TREE REPRESENTATION</th>
<th>TIME USED FOR LINKING A TREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT CHARACTER</td>
<td>TREE DOMAIN</td>
<td>PRIMITIVE</td>
</tr>
<tr>
<td>M</td>
<td>M M M M M M M M M M</td>
<td>1</td>
</tr>
<tr>
<td>n</td>
<td>M M M M M M M M M M</td>
<td>11</td>
</tr>
<tr>
<td>M</td>
<td>M M M M M M M M M M</td>
<td>12</td>
</tr>
<tr>
<td>M</td>
<td>M M M M M M M M M M</td>
<td>1111</td>
</tr>
<tr>
<td>n</td>
<td>M M M M M M M M M M</td>
<td>11112</td>
</tr>
<tr>
<td>M</td>
<td>M M M M M M M M M M</td>
<td>11111</td>
</tr>
<tr>
<td>n</td>
<td>M M M M M M M M M M</td>
<td>121</td>
</tr>
</tbody>
</table>

TIME USED FOR LINKING A TREE: .189 SEC

INPUT CHARACTER IS A
DISTANCE FORM NORMAL A IS 0
TIME USED FOR PARsing: 4.911 SEC

INPUT CHARACTER IS A
DISTANCE FORM NORMAL A IS 2
TIME USED FOR PARSING: 5.154 SEC

BEST AVAILABLE COPY
INPUT CHARACTER

<table>
<thead>
<tr>
<th>MM</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>MM</td>
<td>MM</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

TIME USED FOR LINKING A TREE
186 SEC

INPUT CHARACTER IS A
DISTANCE FROM NORMAL A IS 3
TIME USED FOR PARSING
5.161 SEC

INPUT CHARACTER

<table>
<thead>
<tr>
<th>MM</th>
<th>MM</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>MM</td>
<td>MM</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

TIME USED FOR LINKING A TREE
192 SEC

INPUT CHARACTER IS A
DISTANCE FROM NORMAL A IS 3
TIME USED FOR PARSING
4.924 SEC

BEST AVAILABLE COPY
INPUT CHARACTER

INPUT CHARACTER

TREE REPRESENTATION

TREE REPRESENTATION

TREE DOMAIN PRIMITIVE RANK

TREE DOMAIN PRIMITIVE RANK

TIME USED FOR LINKING A TREE

TIME USED FOR LINKING A TREE

INPUT CHARACTER IS A

INPUT CHARACTER IS C

DISTANCE FROM NORMAL A

DISTANCE FROM NORMAL C

IS 3

IS 0

TIME USED FOR PARSING

TIME USED FOR PARSING

2.669 SEC

2.977 SEC
INPUT CHARACTER

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>3</td>
</tr>
<tr>
<td>1111</td>
<td>2</td>
</tr>
<tr>
<td>11111</td>
<td>1</td>
</tr>
</tbody>
</table>

-tree representation

TIME USED FOR LINKING A TREE
186 SEC

INPUT CHARACTER IS C
DISTANCE FROM NORMAL C IS 1

TIME USED FOR PARSING
2.967 SEC

INPUT CHARACTER

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>111</td>
<td>3</td>
</tr>
<tr>
<td>1111</td>
<td>2</td>
</tr>
<tr>
<td>11111</td>
<td>1</td>
</tr>
</tbody>
</table>

-tree representation

TIME USED FOR LINKING A TREE
193 SEC

INPUT CHARACTER IS C
DISTANCE FROM NORMAL C IS 3

TIME USED FOR PARSING
2.861 SEC

BEST AVAILABLE COPY
INPUT CHARACTER

INPUT CHARACTER

TREE REPRESENTATION

TREE REPRESENTATION

TIME USED FOR LINKING A THEL

TIME USED FOR LINKING A THEL

INPUT CHARACTER IS 'C'

INPUT CHARACTER IS 'C'

DISTANCE FORM NORMAL 'C'

DISTANCE FORM NORMAL 'C'

TIME USED FOR PARSING

TIME USED FOR PARSING

2.824 SEC

2.964 SEC

BEST AVAILABLE COPY
<table>
<thead>
<tr>
<th>INPUT CHARACTER</th>
<th>TREE REPRESENTATION</th>
<th>TIME USED FOR LINKING A TREE</th>
</tr>
</thead>
<tbody>
<tr>
<td>M M M M M M M</td>
<td>1</td>
<td>.190 SEC</td>
</tr>
<tr>
<td>M M</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>M M</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>M M</td>
<td>M M M M M M M M M M</td>
<td></td>
</tr>
<tr>
<td>M M</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>M M M M M M M</td>
<td>1111</td>
<td></td>
</tr>
<tr>
<td>M M</td>
<td>11111</td>
<td></td>
</tr>
<tr>
<td>M M</td>
<td>12111</td>
<td></td>
</tr>
</tbody>
</table>

TIME USED FOR PARSING 5.139 SEC
<table>
<thead>
<tr>
<th>INPUT CHARACTER</th>
<th>TREE REPRESENTATION</th>
<th>TREE DOMAIN PRIMITIVE</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>!1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>112</td>
<td>1111</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1211</td>
<td>12111</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME USED FOR LINKING A TREE</td>
<td>0.199 SEC</td>
<td>TIME USED FOR LINKING A TREE</td>
<td>0.199 SEC</td>
</tr>
<tr>
<td>INPUT CHARACTER IS U</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISTANCE FROM NORMAL U</td>
<td>12.2</td>
<td>DISTANCE FROM NORMAL U</td>
<td>12.2</td>
</tr>
<tr>
<td>TIME USED FOR PARSING</td>
<td>4.335 SEC</td>
<td>TIME USED FOR PARSING</td>
<td>4.335 SEC</td>
</tr>
</tbody>
</table>
INPUT CHARACTER

```
```

INPUT CHARACTER

```
```

THEE REPRESENTATION
TREE DOMAIN PRIMITIVE RANK

```
```

TIME USED FOR LINKING A TREE .197 SEC

```
```

INPUT CHARACTER IS U
DISTANCE FROM NORMAL U IS 4
TIME USED FOR PARSING 6.666 SEC

```
```

BEST AVAILABLE COPY
<table>
<thead>
<tr>
<th>INPUT CHARACTER</th>
<th>INPUT CHARACTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>M M M M M M M M</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

**TREE REPRESENTATION**

<table>
<thead>
<tr>
<th>TREE DOMAIN</th>
<th>PRIMITIVE</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1111</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1112</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11111</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1111111</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**TIME USED FOR LINKING A TREE**

- 192 SEC

**INPUT CHARACTER IS E**

- DISTANCE FROM NORMAL L IS 1
- TIME USED FOR PARSING 9.497 SEC

<table>
<thead>
<tr>
<th>INPUT CHARACTER</th>
<th>INPUT CHARACTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>M M M M M M M M</td>
<td>M M M M M M M M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

**TREE REPRESENTATION**

<table>
<thead>
<tr>
<th>TREE DOMAIN</th>
<th>PRIMITIVE</th>
<th>RANK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1111</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1112</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11111</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>1111111</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**TIME USED FOR LINKING A TREE**

- 195 SEC

**INPUT CHARACTER IS L**

- DISTANCE FROM NORMAL L IS 2
- TIME USED FOR PARSING 6.963 SEC

**BEST AVAILABLE COPY**
INPUT CHARACTER

```
    MMM
   MM MM
  M M M
 M M M M
```

INPUT CHARACTER

```
    MM
   M M
  M M M
 M M M M
```

**TREE REPRESENTATION**

**TREE DOMAIN** | **PRIMITIVE** | **RANK**
-----------------|--------------|---------
      1           |             | 2       
     11           | 6           | 0       
     12           |             | 1       
     111          | 4           | 2       
     1111         | 3           | 0       
     11111        | 3           | 0       

TIME USED FOR LINKING A TREE

.191 SEC

INPUT CHARACTER IS \textit{E}

DISTANCE FORM NORMAL \textit{E}

IS 1

TIME USED FOR PARSING

4.917 SEC
<table>
<thead>
<tr>
<th>Input Character</th>
<th>Input Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tree Representation</th>
<th>Tree Representation</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Tree Domain</th>
<th>Primitive</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>111</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>112</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1121</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Time used for linking a tree: 190 sec

Input Character is 'H'

Distance form normal 'H' is 0

Time used for parsing: 3.761 sec

Input Character is 'M'

Distance form normal 'M' is 1

Time used for parsing: 3.761 sec
INPUT CHARACTER

TREE REPRESENTATION

TREEDomain PRIMITIVE RANK
1 1 1
11 3 2
111 0 1
1111 0 0
1121 1 1
1122 0 0
11211 3 0

TIME USED FOR LINKING A TREELINK
.187 SEC

INPUT CHARACTER IS •
DISTANCE FROM NORMAL • IS 1.

TIME USED FOR PARSING
5.795 SEC

INPUT CHARACTER

TREE REPRESENTATION

TREEDomain PRIMITIVE RANK
1 1 1
11 1 2
111 1 2
1111 1 0
1121 1 0
1122 0 0
11211 3 0

TIME USED FOR LINKING A TREELINK
.193 SEC

INPUT CHARACTER IS •
DISTANCE FROM NORMAL • IS 1.

TIME USED FOR PARSING
5.117 SEC

BEST AVAILABLE COPY
INPUT CHARACTER

M

M

M

M

M

M

M

M

M

TIME USED FOR LINKING A TREE

0.188 SEC

TIME USED FOR PARSING

3.099 SEC

INPUT CHARACTER IS M

DISTANCE FROM NORMAL IS 2

BEST AVAILABLE COPY
APPENDIX E
SYNTHESIS GRAMMARS FOR NETTING AND REPTILE SKIN

E.1. Grammar for Netting (pattern D34)

\[ G_4 = (V_4, r, P_4, S_4) \] over \( \Sigma, r \) and \( G_{S_4} = (V_4, r, P_{S_4}, S_4) \) over \( \Sigma, r \)

\[ V_4 = \{ A_0, 1, \ldots, 9, B_0, 1, \ldots, 9, C_0, 1, \ldots, 9, D_0, 1, \ldots, 9, E_0, 1, \ldots, 9, F_0, 1, \ldots, 9, N_0, 1, \ldots, 6, V_0, 1, \ldots, 3 \} \]

\[ S_4 = \{ A_1, B_0, B_9, C_8, A_7, B_6, C_5, A_4, B_3, C_2, D_1, E_0, E_9, F_8, D_7, E_6, F_5, D_4, E_3, F_2 \} \]

\[ P_4 : \]
\[
\begin{align*}
A_1 + &/ \quad ; \quad / \\
N_0 & A_2 N_0 N_0 N_0
\end{align*}
\]

\[ P_{S_4} : \]
\[
\begin{align*}
A_1 + &/ \quad ; \quad / \\
V_0 & A_2 V_0 V_0 V_0
\end{align*}
\]

\[ A_2 + \]
\[
\begin{align*}
0 + &/ \quad ; \quad / \\
N_1 & A_3 N_1 N_1 N_1
\end{align*}
\]

\[ A_2 + \]
\[
\begin{align*}
0 + &/ \quad ; \quad / \\
N_1 & A_3 N_1 N_1 N_1
\end{align*}
\]

\[ A_3 + \]
\[
\begin{align*}
0 + &/ \quad ; \quad / \\
N_2 & A_4 N_2 N_2 N_2
\end{align*}
\]

\[ A_3 + \]
\[
\begin{align*}
0 + &/ \quad ; \quad / \\
N_2 & A_4 N_2 N_2 N_2
\end{align*}
\]

\[ A_4 + \]
\[
\begin{align*}
0 + &/ \quad ; \quad / \\
N_3 & A_5 N_3 N_3 N_3
\end{align*}
\]

\[ A_4 + \]
\[
\begin{align*}
0 + &/ \quad ; \quad / \\
N_3 & A_5 N_3 N_3 N_3
\end{align*}
\]
\[
\begin{align*}
A_5 & + \quad 0 \quad \quad \quad 0 \quad \quad \quad \quad 0 \quad \quad \quad \quad 0 \\
N_4 & \quad A_6 & N_4 & N_4 & N_4

A_5 & + \quad 0 \quad \quad \quad \quad \quad 0 \quad \quad \quad \quad \quad 0 \\
N_4 & \quad A_6 & N_4 & N_4 & N_4

A_6 & + \quad 0 \quad \quad \quad 0 \quad \quad \quad 0 \quad \quad \quad 0 \\
N_0 & \quad A_7 & N_0 & N_0 & N_0

A_6 & + \quad N_0 \quad 0 \quad \quad \quad 0 \quad \quad \quad \quad \quad 0 \\
N_0 & \quad A_7 & N_0 & N_0 & N_0
\end{align*}
\]
<table>
<thead>
<tr>
<th>D4</th>
<th>1</th>
<th>\ \</th>
<th>1</th>
<th>0.9</th>
<th>\ \</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>N0</td>
<td>D5</td>
<td>N0</td>
<td>N0</td>
<td>N0</td>
<td>N0</td>
<td>N0</td>
</tr>
<tr>
<td>D5</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>N0</td>
<td>D6</td>
<td>N1</td>
<td>N0</td>
<td>N1</td>
<td>N0</td>
<td>N1</td>
</tr>
<tr>
<td>D6</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>N0</td>
<td>D7</td>
<td>N6</td>
<td>N0</td>
<td>N6</td>
<td>N0</td>
<td>N6</td>
</tr>
<tr>
<td>D7</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>N0</td>
<td>D8</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
</tr>
<tr>
<td>D8</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>N0</td>
<td>E0</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
</tr>
<tr>
<td>D9</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>N0</td>
<td>E1</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
</tr>
<tr>
<td>E0</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>N0</td>
<td>E2</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
</tr>
<tr>
<td>E1</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>N0</td>
<td>E3</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
<td>N0</td>
<td>N3</td>
</tr>
<tr>
<td>E2</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>\ \</td>
<td>0</td>
<td>0.9</td>
</tr>
<tr>
<td>N0</td>
<td>E3</td>
<td>V3</td>
<td>N0</td>
<td>V3</td>
<td>N0</td>
<td>V3</td>
</tr>
</tbody>
</table>
\[
\begin{align*}
F_2 &+ 0 & F_2 &+ 0, 0.9 & F_2 &+ 0, 0.1 \\
N_3 & F_3 N_0 & N_3 & N_0 & N_3 & F_3 N_0 & N_3 & N_0 \\
F_3 &+ 0 & F_3 &+ 0, 0.9 & F_3 &+ 0, 0.1 \\
N_3 & F_4 N_0 & N_3 & N_0 & N_3 & F_4 N_0 & N_3 & N_0 \\
F_4 &+ 0 & F_4 &+ 0, 0.9 & F_4 &+ 0, 0.1 \\
N_3 & F_5 N_0 & N_3 & N_0 & N_3 & F_5 N_0 & N_3 & N_0 \\
F_5 &+ 0 & F_5 &+ 0, 0.9 & F_5 &+ 0, 0.1 \\
N_3 & F_6 N_0 & N_3 & N_0 & N_3 & F_6 N_0 & N_3 & N_0 \\
F_6 &+ 0 & F_6 &+ 0, 0.9 & F_6 &+ 0, 0.1 \\
N_3 & F_7 N_0 & N_3 & N_0 & N_3 & F_7 N_0 & N_3 & N_0 \\
F_7 &+ 0 & F_7 &+ 0, 0.9 & F_7 &+ 0, 0.1 \\
N_3 & F_8 N_0 & N_3 & N_0 & N_3 & F_8 N_0 & N_3 & N_0 \\
F_8 &+ 0 & F_8 &+ 0, 0.9 & F_8 &+ 0, 0.1 \\
N_3 & F_9 N_0 & N_3 & N_0 & N_3 & F_9 N_0 & N_3 & N_0 \\
F_9 &+ 0 & F_9 &+ 0, 0.9 & F_9 &+ 0, 0.1 \\
N_3 & D_0 N_0 & N_3 & N_0 & N_3 & D_0 N_0 & N_3 & N_0 \\
D_0 &+ 0 & D_0 &+ 0, 0.9 & D_0 &+ 0, 0.1 \\
N_3 & D_1 N_0 & N_3 & N_0 & N_3 & D_1 N_0 & N_3 & N_0
\end{align*}
\]
\[
\begin{align*}
N_0 & \rightarrow 0 & N_0 & \rightarrow 0, 0.75 & 0, 0.25 \\
& \quad | \quad N_0 & & \quad | \\
N_1 & \rightarrow 0 & V_0 & \rightarrow 1, 0.6 & 1, 0.4 \\
& \quad | \quad N_0 & & \quad | \\
N_2 & \rightarrow 1 & N_1 & \rightarrow 0, 0.5 & 0, 0.2 \\
& \quad | \quad N_1 & & \quad | \\
N_3 & \rightarrow 0 & N_2 & \rightarrow 1, 0.2 & 0, 0.1 \\
& \quad | \quad N_2 & & \quad | \\
N_4 & \rightarrow 0 & N_2 & \rightarrow 1, 1.0 \\
& \quad | \quad N_3 & & \quad | \\
N_5 & \rightarrow 1 & N_3 & \rightarrow 0, 1.0 \\
& \quad | \quad N_2 & & \quad | \\
N_6 & \rightarrow 0 & N_4 & \rightarrow 0, 1.0 \\
& \quad | \quad N_5 & & \quad | \\
N_5 & \rightarrow 1 & N_5 & \rightarrow 0, 0.7 & 0, 0.3 \\
& \quad | \quad N_2 & & \quad | \\
N_6 & \rightarrow 0 & N_5 & \rightarrow 1.0 \\
& \quad | \quad N_5 & & \quad | 
\end{align*}
\]
\[
G_4' = (V_4', r_4', P_4', X_1') \text{ over } \langle \Sigma_4', r_4' \rangle \\
V_4' = \{x_{1,2}, \ldots, 9, 0 \mid y_{1,2}, \ldots, 9, 0 \} \cup \Sigma_4' \\
\Sigma_4' = \{A_1, B_0, B_9, C_8, A_7, B_6, C_5, A_4, B_3, C_2, D_1, E_0, E_9, F_8, D_7, E_6, F_5, D_4, E_3, F_2\} \\
r_4' = \{0, 1, 2\} \\
P_4': \\
X_1' \rightarrow A_1 \mid \bar{A}_1 \mid A_1 \\
\quad \downarrow y_1 \quad \downarrow x_2 \quad \downarrow x_2 \\
X_2' \rightarrow B_0 \mid \bar{B}_0 \\
\quad \downarrow x_3 \\
X_3' \rightarrow B_9 \mid \bar{B}_9 \\
\quad \downarrow x_4 \\
X_4' \rightarrow C_8 \mid \bar{C}_8 \\
\quad \downarrow x_5 \\
X_5' \rightarrow A_7 \mid \bar{A}_7
\[
\begin{align*}
    x_6 &\rightarrow B_6 \quad \Rightarrow \quad | \quad B_6 \\
    y_6 &\rightarrow x_7 \\
    x_7 &\rightarrow C_5 \quad \Rightarrow \quad | \quad C_5 \\
    x_8 &\rightarrow \quad | \\
    x_8 &\rightarrow A_4 \quad \Rightarrow \quad | \quad A_4 \\
    x_9 &\rightarrow \quad | \\
    x_9 &\rightarrow B_3 \quad \Rightarrow \quad | \quad B_3 \\
    x_0 &\rightarrow \quad | \\
    x_0 &\rightarrow C_2 \quad \Rightarrow \quad | \quad C_2 \\
    x_1 &\rightarrow \quad | \\
    y_1 &\rightarrow D_1 \quad \Rightarrow \quad | \quad D_1 \\
    x_6 &\rightarrow y_2 \\
    y_2 &\rightarrow E_0 \quad \Rightarrow \quad | \quad E_0 \\
    y_3 &\rightarrow \quad | \\
    y_3 &\rightarrow E_9 \quad \Rightarrow \quad | \quad E_9 \\
    y_4 &\rightarrow F_8 \quad \Rightarrow \quad | \quad F_8 \\
    y_5 &\rightarrow D_7 \quad \Rightarrow \quad | \quad D_7 \\
    y_5 &\rightarrow \quad .
\end{align*}
\]
E.2. Grammar for Reptile Skin (pattern D22)

\[ G_5 = (V_5, r, P_5, S_5) \text{ over } \Sigma, r \]

\[ V_5 = S_5 \cup \Sigma \cup \{N_0,1,\ldots,9, V_0,1,\ldots,5\} \]

\[ S_5 = \{X_1 | X \in \{A,B,C,D,E,F,G,H,I\}, i \in \{1,2,\ldots,9\}\} \]

\[ P_5: \]

\[ A_1 + \begin{array}{c}
\end{array} \begin{array}{c}
\end{array} \begin{array}{c}
\end{array} \begin{array}{c}
\end{array} \]

\[ A_2 + \begin{array}{c}
\end{array} \begin{array}{c}
\end{array} \begin{array}{c}
\end{array} \begin{array}{c}
\end{array} \]

\[ A_3 + \begin{array}{c}
\end{array} \begin{array}{c}
\end{array} \begin{array}{c}
\end{array} \begin{array}{c}
\end{array} \]
\[ \begin{align*}
C_4 + & 0 \\
N_7 & C_5 V_2
\end{align*} \]

\[ \begin{align*}
C_5 + & 1 \\
V_3 & C_6 N_4
\end{align*} \]

\[ \begin{align*}
C_6 + & 1 \\
N_7 & C_7 V_2
\end{align*} \]

\[ \begin{align*}
C_7 + & 1 \\
V_4 & C_8 V_0
\end{align*} \]

\[ \begin{align*}
C_8 + & 1 \\
N_5 & C_9 V_0
\end{align*} \]

\[ \begin{align*}
C_9 + & 1 \\
V_5 & C_1 V_0
\end{align*} \]

\[ \begin{align*}
D_1 + & 1 \\
V_3 & D_2 V_0
\end{align*} \]

\[ \begin{align*}
D_2 + & 1 \\
N_1 & D_3 V_0
\end{align*} \]

\[ \begin{align*}
D_3 + & 1 \\
N_7 & D_4 V_0
\end{align*} \]
\[\begin{align*}
D_4 & \rightarrow \begin{array}{c}
0 \\
V_1 \ D_5 \ V_1
\end{array} & D_4 & \rightarrow \begin{array}{c}
0 \\
V_1 \ V_1
\end{array} \\
D_5 & \rightarrow \begin{array}{c}
1 \\
V_1 \ D_6 \ N_1
\end{array} & D_5 & \rightarrow \begin{array}{c}
1 \\
V_1 \ N_1
\end{array} \\
D_6 & \rightarrow \begin{array}{c}
0 \\
V_1 \ D_7 \ V_1
\end{array} & D_6 & \rightarrow \begin{array}{c}
0 \\
V_1 \ V_1
\end{array} \\
D_7 & \rightarrow \begin{array}{c}
1 \\
N_7 \ D_8 \ V_0
\end{array} & D_7 & \rightarrow \begin{array}{c}
1 \\
N_7 \ V_0
\end{array} \\
D_8 & \rightarrow \begin{array}{c}
1 \\
N_1 \ D_9 \ V_0
\end{array} & D_8 & \rightarrow \begin{array}{c}
1 \\
N_1 \ V_0
\end{array} \\
D_9 & \rightarrow \begin{array}{c}
1 \\
V_3 \ D_1 \ V_0
\end{array} & D_9 & \rightarrow \begin{array}{c}
1 \\
V_3 \ V_0
\end{array} \\
E_1 & \rightarrow \begin{array}{c}
1 \\
N_6 \ E_2 \ V_0
\end{array} & E_1 & \rightarrow \begin{array}{c}
1 \\
N_6 \ V_0
\end{array} \\
E_2 & \rightarrow \begin{array}{c}
1 \\
N_4 \ E_3 \ V_0
\end{array} & E_2 & \rightarrow \begin{array}{c}
1 \\
N_4 \ V_0
\end{array} \\
E_3 & \rightarrow \begin{array}{c}
0 \\
V_1 \ E_4 \ V_0
\end{array} & E_3 & \rightarrow \begin{array}{c}
0 \\
V_1 \ V_0
\end{array}
\end{align*}\]
<table>
<thead>
<tr>
<th>$F_4$</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_5$</td>
<td>1</td>
</tr>
<tr>
<td>$F_6$</td>
<td>1</td>
</tr>
<tr>
<td>$F_7$</td>
<td>0</td>
</tr>
<tr>
<td>$F_8$</td>
<td>0</td>
</tr>
<tr>
<td>$F_9$</td>
<td>0</td>
</tr>
<tr>
<td>$G_1$</td>
<td>1</td>
</tr>
<tr>
<td>$G_2$</td>
<td>0</td>
</tr>
<tr>
<td>$G_3$</td>
<td>1</td>
</tr>
</tbody>
</table>
\begin{align*}
1_4 & + \begin{array}{c}
\v_0 \\
\v_5 \\
\v_2 
\end{array} \quad ; \quad 1 \\
1_5 & + \begin{array}{c}
\v_0 \\
\v_6 \\
\v_2 
\end{array} \quad ; \quad 1 \\
1_6 & + \begin{array}{c}
\v_0 \\
\v_7 \\
\v_2 
\end{array} \quad ; \quad 1 \\
1_7 & + \begin{array}{c}
\v_0 \\
\v_8 \\
\n_5 
\end{array} \quad ; \quad 1 \\
1_8 & + \begin{array}{c}
\n_5 \\
\n_9 \\
\n_4 
\end{array} \quad ; \quad 1 \\
1_9 & + \begin{array}{c}
\n_7 \\
\v_1 \\
\v_5 
\end{array} \quad ; \quad 1 \\
\n_0 & + \begin{array}{c}
\n_0 
\end{array} \quad ; \quad 0 \\
\n_1 & + \begin{array}{c}
\n_0 
\end{array} \quad ; \quad 1 \\
\n_2 & + \begin{array}{c}
\n_1 
\end{array} \quad ; \quad 0
\end{align*}
\[
\begin{array}{cccccc}
0 & & & & & 271 \\
v_1 & \rightarrow & | & & & v_0 \\
& & 0 & & & \\
n_3 & \rightarrow & | & n_2 & & n_5 \\
& & 0 & & & \\
n_4 & \rightarrow & | & n_3 & & n_6 \\
& & 1 & & & \\
n_5 & \rightarrow & | & n_1 & & n_3 \\
& & 0 & & & \\
n_6 & \rightarrow & | & n_5 & & n_8 \\
& & 0 & & & \\
n_7 & \rightarrow & | & n_6 & & \\
& & 1 & & & \\
n_8 & \rightarrow & | & n_2 & & \\
& & 0 & & & \\
n_9 & \rightarrow & | & n_8 & & \\
& & 1 & & & \\
v_0 & \rightarrow & | & \vdots & & 1 \\
& & v_0 & & & \\
\end{array}
\]
APPENDIX F

DISCRIMINATION GRAMMARS FOR PATTERN D22, D34, D38, and D68

F.1. Grammar for Synthesis and Discrimination of Water Pattern (D38)

\[ G_6 = (V_6, R, P_6, S_6) \] over \( \Sigma, r \)

\[ V_6 = S_6 \cup (N_0, 1, \ldots, 9, V_0, 1, \ldots, 5) \cup \Sigma \]

\[ S_6 = \{ A_0, 1, B_1, 2, \ldots, \}, C_1, 2, \ldots, 6, D_1, 2, \ldots, 7, E_1, 2, \ldots, 5, F_1, 2, \ldots, 5, G_1, 2, 3, H_1, 2, 3, I_1, J_1 \} \]

\[ P_6: \]

\[ A_1 + \]

\[ N_0 A_1 N_0, 0.1; N_0 B_1 N_0, 0; N_0 B_2 N_0, 0.1; \]

\[ N_0 C_1 N_0, 0.1; N_0 D_1 N_0, 0; N_0 E_1 N_0, 0.1; \]

\[ N_0 F_1 N_0, 0.1; N_0 G_1 N_0, 0; N_0 H_1 N_0, 0.1; \]

\[ N_0 I_1 N_0, 0.1; N_0 J_1 N_0, 0; N_0 C_6 N_0, 0; \]

\[ N_0 D_7 N_0, 0; N_0 N_0, 0.1; \]
\[
A_0 + \begin{array}{c}
\begin{array}{c}
1 \\
B_7 V_0
\end{array} \\
V_0
\end{array}, 0; \quad \begin{array}{c}
1 \\
B_6 V_0
\end{array}, 0; \quad \begin{array}{c}
1 \\
C_3 V_0
\end{array}, 0;
\]
\[
\begin{array}{c}
1 \\
C_4 V_0
\end{array}, 0; \quad \begin{array}{c}
1 \\
D_3 V_0
\end{array}, 0; \quad \begin{array}{c}
1 \\
D_4 V_0
\end{array}, 0;
\]
\[
\begin{array}{c}
1 \\
D_5 V_0
\end{array}, 0; \quad \begin{array}{c}
1 \\
E_4 V_0
\end{array}, 0; \quad \begin{array}{c}
1 \\
F_4 V_0
\end{array}, 0;
\]
\[
\begin{array}{c}
1 \\
G_3 V_0
\end{array}, 0; \quad \begin{array}{c}
1 \\
H_3 V_0
\end{array}, 0; \quad \begin{array}{c}
1 \\
A_0 V_0
\end{array}, 0;
\]
\[
B_1 + \begin{array}{c}
\begin{array}{c}
0 \\
B_1 N_0
\end{array} \\
N_4
\end{array}, 0; \quad \begin{array}{c}
0 \\
B_3 N_0
\end{array}, 0; \quad \begin{array}{c}
0 \\
A_0 N_0
\end{array}, 0;
\]
\[
\begin{array}{c}
0 \\
C_6 N_0
\end{array}, 0; \quad \begin{array}{c}
0 \\
C_4 N_0
\end{array}, 0;
\]
\[
B_2 + \begin{array}{c}
\begin{array}{c}
0 \\
B_2 N_0
\end{array} \\
N_7
\end{array}, 0.4; \quad \begin{array}{c}
1 \\
B_4 N_0
\end{array}, 0.3; \quad \begin{array}{c}
0 \\
A_0 N_0
\end{array}, 0.2;
\]
\[
\begin{array}{c}
0 \\
D_7 N_0
\end{array}, 0; \quad \begin{array}{c}
0 \\
N_7 N_0
\end{array}, 0.1;
\]
\[ B_3 + \begin{array}{c} 0 \\ \hline V_1 B_3 N_0 \\ \hline 0 \end{array}, 0; \quad \begin{array}{c} 0 \\ \hline V_1 B_1 N_0 \\ \hline 0 \end{array}, 0; \quad \begin{array}{c} 0 \\ \hline V_1 B_5 N_0 \\ \hline 0 \end{array}, 0; \]

\[ B_4 + \begin{array}{c} 0 \\ \hline V_0 B_4 N_0 \\ \hline 0 \end{array}, 0.4; \quad \begin{array}{c} 0 \\ \hline V_0 B_2 N_0 \\ \hline 0 \end{array}, 0.2; \quad \begin{array}{c} 0 \\ \hline V_0 B_6 N_0 \\ \hline 0 \end{array}, 0.15; \]

\[ B_5 + \begin{array}{c} 1 \\ \hline V_0 B_5 N_0 \\ \hline 1 \end{array}, 0; \quad \begin{array}{c} 1 \\ \hline V_0 B_3 N_0 \\ \hline 1 \end{array}, 0; \quad \begin{array}{c} 1 \\ \hline V_0 B_7 N_0 \\ \hline 1 \end{array}, 0; \]

\[ B_6 + \begin{array}{c} 1 \\ \hline V_0 B_6 N_1 \\ \hline 1 \end{array}, 0.4; \quad \begin{array}{c} 1 \\ \hline V_0 B_4 N_1 \\ \hline 1 \end{array}, 0.3; \quad \begin{array}{c} 1 \\ \hline V_0 A_0 N_1 \\ \hline 1 \end{array}, 0; \]

\[ \begin{array}{c} 1 \\ \hline V_0 D_2 N_1 \\ \hline 1 \end{array}, 0.2; \quad \begin{array}{c} 1 \\ \hline V_0 H_3 N_1 \\ \hline 1 \end{array}, 0; \quad \begin{array}{c} 1 \\ \hline V_0 N_1 \\ \hline 1 \end{array}, 0.1; \]
\[
B_7 + \frac{1}{V_0 B_7 N_5}, 0; \quad \frac{1}{V_0 B_5 N_5}, 0; \quad \frac{1}{V_0 A_0 N_1}, 0;
\]

\[
c_1 + \frac{1}{N_6 C_1 N_0}, 0.4; \quad \frac{1}{N_6 A_1 N_0}, 0.2; \quad \frac{1}{N_6 C_2 N_0}, 0.3;
\]

\[
c_2 + \frac{1}{V_2 C_2 N_0}, 0.4; \quad \frac{1}{V_2 C_1 N_0}, 0.2; \quad \frac{1}{V_2 C_3 N_0}, 0.15;
\]

\[
c_3 + \frac{1}{V_2 C_3 N_5}, 0.4; \quad \frac{1}{V_2 C_2 N_5}, 0.3; \quad \frac{1}{V_2 E_2 N_0}, 0.3;
\]

\[
1, 0.1; \quad 1, 0.01;
\]
\[
\begin{align*}
C_4 + &\quad 1 \quad 0; \quad 1 \quad 0; \quad 1 \quad 0; \\
&\quad v_4 \quad c_4 \quad v_0; \quad v_4 \quad c_5 \quad v_0; \quad v_4 \quad a_0 \quad v_0; \\
&\quad 1 \quad 0; \quad 1 \quad 0; \\
&\quad v_4 \quad e_3 \quad v_0; \quad v_4 \quad v_0; \\
C_5 + &\quad 0 \quad 0; \quad 0 \quad 0; \quad 0 \quad 0; \\
&\quad n_4 \quad c_5 \quad v_0; \quad n_4 \quad c_4 \quad v_0; \quad n_4 \quad c_6 \quad v_0; \\
&\quad 0 \quad 0; \quad 0 \quad 0; \quad 0 \quad 0; \\
&\quad n_4 \quad e_4 \quad v_0; \quad n_4 \quad g_2 \quad v_0; \quad n_4 \quad v_0; \\
C_6 + &\quad 0 \quad 0; \quad 0 \quad 0; \quad 0 \quad 0; \\
&\quad n_4 \quad c_6 \quad n_7; \quad n_4 \quad c_5 \quad n_7; \quad n_4 \quad e_5 \quad n_7; \\
&\quad 0 \quad 0; \quad 0 \quad 0; \\
&\quad n_4 \quad a_1 \quad n_7; \quad n_4 \quad n_7; \\
D_1 + &\quad 0 \quad 0.4; \quad 0 \quad 0.2; \quad 0 \quad 0.15; \\
&\quad n_5 \quad d_1 \quad n_0; \quad n_5 \quad a_1 \quad n_0; \quad n_5 \quad d_2 \quad n_0; \\
&\quad 0 \quad 0.15; \quad 0 \quad 0.1; \\
&\quad n_5 \quad b_4 \quad n_0; \quad n_5 \quad n_0;
\end{align*}
\]
$D_2 + \begin{array}{c}
N_5 \quad D_2 \quad N_1 \\
1 \quad 0.4 \\
N_5 \quad D_1 \quad N_1 \\
1 \quad 0.1 \\
N_5 \quad D_3 \quad N_1 \\
1 \quad 0.2 \\
\end{array}$

$D_3 + \begin{array}{c}
N_5 \quad D_3 \quad V_2 \\
1 \quad 0.4 \\
N_5 \quad D_2 \quad V_2 \\
1 \quad 0.15 \\
N_5 \quad D_4 \quad V_2 \\
1 \quad 0 \\
\end{array}$

$D_4 + \begin{array}{c}
N_7 \quad D_4 \quad V_0 \\
1 \quad 0 \\
N_7 \quad D_3 \quad V_0 \\
1 \quad 0 \\
N_7 \quad A_0 \quad V_0 \\
1 \quad 0 \\
\end{array}$

$D_5 + \begin{array}{c}
N_7 \quad D_5 \quad V_0 \\
1 \quad 0 \\
N_7 \quad D_6 \quad V_0 \\
1 \quad 0 \\
N_7 \quad F_3 \quad V_0 \\
1 \quad 0 \\
\end{array}$
\[ E_3 + \frac{1}{N_1 E_3 V_0}, 0.4; \frac{1}{N_1 E_2 V_0}, 0.3; \frac{1}{N_1 G_2 V_0}, 0.2; \]

\[ E_4 + \frac{1}{V_2 E_4 V_0}, 0; \frac{1}{V_2 E_5 V_0}, 0; \frac{1}{V_2 A_0 V_0}, 0; \]

\[ E_5 + \frac{1}{V_2 E_5 N_7}, 0; \frac{1}{V_2 E_4 N_7}, 0; \frac{1}{V_2 G_3 N_7}, 0; \]

\[ F_1 + \frac{1}{N_0 F_1 N_1}, 0.4; \frac{1}{N_0 F_2 N_1}, 0.15; \frac{1}{N_0 A_1 N_1}, 0.2; \]

\[ \frac{1}{N_0 D_2 N_1}, 0.15; \frac{1}{N_0 N_1}, 0.1 \]
\[
\begin{align*}
F_2 + & \quad 1 \\
& \quad | \\
N_0 & \quad F_2 \quad V_2 , 0.4 \\
& \quad | \\
N_0 & \quad F_3 \quad V_2 , 0.15 \\
& \quad | \\
N_0 & \quad F_1 \quad V_2 , 0.1 \\
& \quad | \\
N_0 & \quad D_3 \quad V_2 , 0.15 \\
& \quad | \\
N_0 & \quad H_1 \quad V_2 , 0.1 \\
& \quad | \\
N_0 & \quad V_2 , 0.1 \\
F_3 + & \quad 1 \\
& \quad | \\
N_0 & \quad F_3 \quad V_0 , 0.4 \\
& \quad | \\
N_0 & \quad F_2 \quad V_0 , 0.3 \\
& \quad | \\
N_0 & \quad D_4 \quad V_0 , 0 \\
& \quad | \\
N_0 & \quad H_2 \quad V_0 , 0.2 \\
& \quad | \\
N_0 & \quad D_5 \quad V_0 , 0.1 \\
F_4 + & \quad 0 \\
& \quad | \\
V_0 & \quad F_4 \quad V_1 , 0 \\
& \quad | \\
V_0 & \quad A_0 \quad V_1 , 0 \\
& \quad | \\
V_0 & \quad D_6 \quad V_1 , 0 \\
& \quad | \\
V_0 & \quad V_1 , 0 \\
F_5 + & \quad 0 \\
& \quad | \\
V_0 & \quad F_5 \quad N_4 , 0 \\
& \quad | \\
V_0 & \quad F_4 \quad N_4 , 0 \\
& \quad | \\
V_0 & \quad D_7 \quad N_4 , 0 \\
& \quad | \\
V_0 & \quad H_3 \quad N_4 , 0 \\
& \quad | \\
V_0 & \quad N_4 , 0
\end{align*}
\]
\[ G_1 + 0, 0.4; \]
\[ N_0 G_1 N_5 \]
\[ 0, 0.15; \]
\[ N_0 G_2 N_5 \]
\[ 0, 0.15; \]
\[ N_0 E_2 N_5 \]
\[ 0, 0.2; \]
\[ N_0 A_1 N_5 \]
\[ 0, 0.1; \]
\[ N_0 N_5 \]
\[ G_2 + 0, 0.4; \]
\[ N_0 G_2 V_0 \]
\[ 0, 0.1; \]
\[ N_0 G_1 V_0 \]
\[ 0, 0.3; \]
\[ N_0 E_3 V_0 \]
\[ 0, 1; \]
\[ N_0 I_1 V_0 \]
\[ 0, 0; \]
\[ N_0 C_5 V_0 \]
\[ 0, 0.1; \]
\[ N_0 V_0 \]
\[ G_3 + 1, 0; \]
\[ V_0 G_3 N_7 \]
\[ 1, 0; \]
\[ V_0 A_0 N_7 \]
\[ 1, 0; \]
\[ V_0 B_5 N_7 \]
\[ 1, 0; \]
\[ V_0 E_5 N_7 \]
\[ 1, 0; \]
\[ V_0 N_7 \]
\[ H_1 + 0, 0.4; \]
\[ N_0 H_1 N_6 \]
\[ 0, 0.15; \]
\[ N_0 H_2 N_6 \]
\[ 0, 0.2; \]
\[ N_0 A_1 N_6 \]
\[ 0, 0.15; \]
\[ N_0 F_2 N_6 \]
\[ 0, 0.1; \]
\[ N_0 N_6 \]
\begin{align*}
N_0 + & | & 0.9 & ; & 0 & , & 0.1 \\
N_0 & & & & & & \\
N_1 + & | & 0.9 & ; & 1 & , & 0.1 \\
N_0 & & & & & & \\
N_2 + & | & 0.1 & , & 0.9 & ; & 1 & , & 0.1 \\
N_1 & & & & & & \\
N_3 + & | & 0.1 & , & 0.9 & ; & 1 & , & 0.1 \\
N_2 & & & & & & \\
N_4 + & | & 0.1 & , & 1 & , & 1 & , & 0.1 \\
N_3 & & & & & & \\
N_5 + & | & 0.1 & , & 1 & , & 1 & , & 0.1 \\
N_4 & & & & & & \\
N_6 + & | & 0.1 & , & 1 & , & 1 & , & 0.1 \\
N_5 & & & & & & \\
N_7 + & | & 0.1 & , & 1 & , & 1 & , & 0.1 \\
N_6 & & & & & & \\
N_8 + & | & 0.1 & , & 1 & , & 1 & , & 0.1 \\
N_7 & & & & & & \\
N_9 + & | & 0.1 & , & 1 & , & 1 & , & 0.1 \\
N_8 & & & & & & \\
V_0 + & | & 0.9 & ; & 1 & , & 0.1 \\
V_0 & & & & & & \\
V_1 + & | & 0.1 & , & 1 & , & 1 & , & 0.1 \\
V_0 & & & & & & \\
V_2 + & | & 0.1 & , & 1 & , & 1 & , & 0.1 \\
V_5 & & & & & & 
\end{align*}
1

\( v_3 \rightarrow | \quad 1 \quad v_4 \rightarrow | \quad 1 \)

\( N_6 \quad N_3 \)

1

\( v_5 \rightarrow | \quad 1 \)

\( N_8 \)
F.2. Discrimination Grammar for Pattern D22

\[ G_{22} = (V_{22}, r, P_{22}, S_{22}) \text{ over } <\Sigma, r> \]

\[ V_{22} = S_{22} \cup \{N_0, 1, \ldots, 9; V_0, 1, \ldots, 5\} \cup \Sigma \]

\[ S_{22} = A_0, 1, \ldots, 9; B_1, 2, \ldots, 8; C_1, 2, \ldots, 7; D_1, 2, \ldots, 6; E_1, 2, \ldots, 6; F_1, 2, \ldots, 6; G_1, 2, \ldots, 6; H_1, 2, \ldots, 6; I_1, 2, 3 \]

\[ P_{22} = \]

\[ A_1 + \]

\[ V_2 A_1 V_0 \]

\[ V_2 B_1 V_0 \]

\[ V_2 C_1 V_0 \]

\[ V_2 A_0 V_0 \]

\[ V_2 V_0 \]

\[ A_2 + \]

\[ V_5 A_2 V_0 \]

\[ V_5 B_1 V_0 \]

\[ V_5 B_2 V_0 \]

\[ V_5 A_0 V_0 \]

\[ V_5 V_0 \]

\[ A_3 + \]

\[ V_3 A_3 V_0 \]

\[ V_3 B_2 V_0 \]

\[ V_3 B_3 V_0 \]

\[ V_3 C_1 V_0 \]

\[ V_3 C_3 V_0 \]

\[ V_3 V_0 \]
\[ \begin{align*}
A_4 & \rightarrow \\
v_1 & A_4 v_0 \\
1 & \\
A_5 & \rightarrow \\
v_0 & A_5 v_0 \\
0 & \\
A_6 & \rightarrow \\
v_0 & A_6 v_1 \\
1 & \\
A_7 & \rightarrow \\
v_0 & A_7 v_3 \\
1 & \\
\end{align*} \]

\[ \begin{align*}
A_4 & \rightarrow \\
v_1 & B_3 v_0 \\
1 & \\
A_5 & \rightarrow \\
v_0 & B_4 v_0 \\
0 & \\
A_6 & \rightarrow \\
v_0 & B_5 v_1 \\
1 & \\
A_7 & \rightarrow \\
v_0 & B_6 v_3 \\
1 & \\
\end{align*} \]
$B_3 \rightarrow \begin{array}{c}
\text{1} \\
\text{N}_7 \text{A}_3 \text{V}_0
\end{array}$

$B_4 \rightarrow \begin{array}{c}
\text{1} \\
\text{N}_7 \text{A}_4 \text{V}_0
\end{array}$

$B_5 \rightarrow \begin{array}{c}
\text{1} \\
\text{N}_7 \text{A}_5 \text{V}_0
\end{array}$

$B_6 \rightarrow \begin{array}{c}
\text{1} \\
\text{N}_7 \text{A}_6 \text{N}_7
\end{array}$
\( E_6 + \) \\
\( V_0 \quad C_5 \quad N_0 \) \\
\( F_1 + \) \\
\( N_0 \quad F_3 \quad V_1 \) \\
\( F_3 + \) \\
\( N_0 \quad F_4 \quad V_1 \) \\
\( F_4 + \) \\
\( N_0 \quad C_4 \quad V_1 \) \\
\( F_2 + \) \\
\( V_1 \quad F_5 \quad N_0 \) \\
\( F_5 + \) \\
\( V_1 \quad F_6 \quad N_0 \) \\
\( F_6 + \) \\
\( V_1 \quad C_4 \quad N_0 \) \\
\( G_1 + \) \\
\( N_0 \quad G_3 \quad N_7 \) \\
\( G_3 + \) \\
\( N_0 \quad G_4 \quad N_7 \)
\[ \begin{align*}
G_4 & \rightarrow O \\
\ & \quad \downarrow \\
\ & \quad N_0 \quad G_5 \quad N_7 \\
\ & \quad O \\
\ & \quad N_0 \quad N_7 \\
\ & \quad O \\
G_5 & \rightarrow O \\
\ & \quad \downarrow \\
\ & \quad N_7 \quad G_6 \quad N_0 \\
\ & \quad O \\
\ & \quad N_7 \quad C_3 \quad N_0 \\
\ & \quad O \\
G_6 & \rightarrow O \\
\ & \quad \downarrow \\
\ & \quad N_7 \quad C_3 \quad N_0 \\
\ & \quad O \\
H_1 & \rightarrow O \\
\ & \quad \downarrow \\
\ & \quad N_0 \quad H_3 \quad N_4 \\
\ & \quad O \\
\ & \quad N_0 \quad C_6 \quad N_4 \\
\ & \quad O \\
H_3 & \rightarrow O \\
\ & \quad \downarrow \\
\ & \quad N_0 \quad H_4 \quad N_4 \\
\ & \quad O \\
\ & \quad N_0 \quad C_6 \quad N_4 \\
\ & \quad O \\
H_4 & \rightarrow O \\
\ & \quad \downarrow \\
\ & \quad N_0 \quad C_6 \quad N_4 \\
\ & \quad O \\
H_2 & \rightarrow O \\
\ & \quad \downarrow \\
\ & \quad N_4 \quad H_5 \quad N_0 \\
\ & \quad O \\
\ & \quad N_4 \quad C_2 \quad N_0 \\
\ & \quad O \\
H_5 & \rightarrow O \\
\ & \quad \downarrow \\
\ & \quad N_4 \quad H_6 \quad N_0 \\
\ & \quad O \\
\ & \quad N_4 \quad C_2 \quad N_0 \\
\ & \quad O \\
\end{align*} \]
\begin{align*}
N_0 & \rightarrow | \; 0 \\
1 & \\
N_1 & \rightarrow | \; 1 \\
N_2 & \rightarrow | \\
0 & \rightarrow | \; 0 \\
0 & \rightarrow | \\
N_6 & \rightarrow | \\
0 & \rightarrow | \\
N_8 & \rightarrow | \\
1 & \\
v_0 & \rightarrow | \; 1 \\
v_1 & \rightarrow | \\
v_2 & \rightarrow | \\
v_0 & \\
0 & \rightarrow | \\
N_5 & \\
1 & \\
N_2 & \\
N_1 & \\
N_6 & \\
N_5 & \\
N_7 & \\
N_6 & \\
N_9 & \\
N_8 & \\
1 & \\
N_0 &
\end{align*}
F.3. **Discrimination Grammar for Pattern D34**

\[ G_{34} = (V_{34}, r, P_{34}, S_{34}) \] over \( \Sigma, r \)

\[ V_{34} = S \cup \Sigma \cup \{ A_0, B_0, C_0, D_0, E_0, F_0 \} \]

\[ S_{34} = \{ A_1, \ldots, B_0, C_0, D_0, E_0, F_0, 0, 1, \ldots, 9, 6 \} \]

\[
\begin{align*}
P_{34} : & \quad A_1 + \quad \begin{array}{c} \ \ 0 \\ N_1 A_1 N_1 \end{array} & & \begin{array}{c} \ \ 1 \\ N_1 A_1 N_1 \end{array} & & \begin{array}{c} \ \ 1 \\ N_1 N_1 \end{array} \\
\end{align*}
\]

\[
\begin{align*}
P_{34} : & \quad A_2 + \quad \begin{array}{c} \ \ 0 \\ N_2 A_3 N_2 \end{array} & & \begin{array}{c} \ \ 1 \\ N_2 A_3 N_2 \end{array} & & \begin{array}{c} \ \ 1 \\ N_2 N_2 \end{array} \\
\end{align*}
\]

\[
\begin{align*}
P_{34} : & \quad A_3 + \quad \begin{array}{c} \ \ 0 \\ N_3 A_4 N_3 \end{array} & & \begin{array}{c} \ \ 1 \\ N_3 A_4 N_3 \end{array} & & \begin{array}{c} \ \ 1 \\ N_2 N_3 \end{array} \\
\end{align*}
\]

\[
\begin{align*}
P_{34} : & \quad A_4 + \quad \begin{array}{c} \ \ 0 \\ N_4 A_5 N_4 \end{array} & & \begin{array}{c} \ \ 1 \\ N_3 N_4 \end{array} & & \begin{array}{c} \ \ 1 \\ N_2 N_3 \end{array} & & \begin{array}{c} \ \ 1 \\ N_2 N_3 \end{array} \\
\end{align*}
\]

\[
\begin{align*}
P_{34} : & \quad A_5 + \quad \begin{array}{c} \ \ 0 \\ N_0 A_5 N_0 \end{array} & & \begin{array}{c} \ \ 1 \\ N_0 A_5 N_0 \end{array} & & \begin{array}{c} \ \ 1 \\ N_0 N_0 \end{array} & & \begin{array}{c} \ \ 1 \\ N_4 N_4 \end{array} \\
\end{align*}
\]

\[
\begin{align*}
P_{34} : & \quad A_6 + \quad \begin{array}{c} \ \ 0 \\ N_3 A_6 N_3 \end{array} & & \begin{array}{c} \ \ 1 \\ N_3 N_3 \end{array} & & \begin{array}{c} \ \ 1 \\ N_2 N_2 \end{array} \\
\end{align*}
\]

\[
\begin{align*}
P_{34} : & \quad A_7 + \quad \begin{array}{c} \ \ 0 \\ N_2 A_8 N_2 \end{array} & & \begin{array}{c} \ \ 1 \\ N_2 N_2 \end{array} \\
\end{align*}
\]
\begin{align*}
B_9 + &\quad 0 \quad ; \quad 0 \\
&\quad N_1 B_0 N_3 \\
B_0 + &\quad 1 \quad ; \quad 1 \\
&\quad N_0 B_1 N_2 \\
C_1 + &\quad 0 \quad ; \quad 0 \\
&\quad N_0 C_1 N_6 \\
C_2 + &\quad 1 \quad ; \quad 1 \\
&\quad N_0 C_3 N_4 \\
C_3 + &\quad 0 \quad ; \quad 0 \\
&\quad N_1 C_4 N_0 \\
C_4 + &\quad 0 \quad ; \quad 0 \\
&\quad N_2 C_5 N_0 \\
C_5 + &\quad 0 \quad ; \quad 0 \\
&\quad N_3 C_6 N_0 \\
C_6 + &\quad 0 \quad ; \quad 0 \\
&\quad N_7 C_6 N_0
\end{align*}
F.4. Discrimination Grammar for Pattern D68

\[ G_{68} = (V_{68}, \Gamma, P_{68}, S_{68}) \] over \( \Sigma, \Gamma \)

\[ V_{68} = \{A_1, A_2, A_3, N_0, N_1, V_0, V_1\} \cup \Sigma \]

\[ S_{68} = \{A_1, A_2, A_3\} \]

\[ P_{68}: \]

\[ A_1 + \]

\[ A_2 + \]

\[ A_3 + \]
APPENDIX G

ALGORITHMS FOR THE COMPUTATION OF THE DISTANCE BETWEEN TWO STRINGS, AND THE MINIMUM SPANNING TREE OF A SET OF PATTERNS

G.1. The Distance Between Two Strings

Algorithm G.1

Input: Two strings \( x = a_1, a_2, \ldots, a_n \), \( y = b_1, b_2, \ldots, b_m \)

Output: \( d(x, y) \)

Method:

Step 1. \( D(0,0) = 0 \)

Step 2. \( \text{DO } i = 1, n \)
\( \text{DO } j = 1, m \)
\( D(i, j) = D(i-1, j) + 1 \)

Step 3. \( \text{DO } i = 1, n \)
\( \text{DO } j = 1, m \)
\( e_1 = D(i-1, j-1) + 1 \)
\( \text{if } a_i = b_j, \text{ or } e_1 = D(i-1, j-1) \text{ if } a_i \neq b_j \)
\( e_2 = D(i-1, j) + 1 \)
\( e_3 = D(i, j-1) + 1 \)
\( D(i, j) = \min(e_1, e_2, e_3) \)

Step 4. \( d(x, y) = D(n, m), \text{ exit} \)

G.2. The Minimum Spanning Tree

Algorithm G.2

Input: \( X = \{x_1, x_2, \ldots, x_n\} \), a set of sentences

Output: The minimum spanning tree of \( X \)
Method:

**Step 1.** Assume that there are n nodes

**Step 2.** Compute \(d(x_i,x_j)\) for all \(i, j\). Let \(d(x_i,x_j)\) be the length of the arc connected node \(i\) and \(j\), and denote as \(d(i,j)\).

**Step 3.** List all arc \((i,j)\) in the order of increasing \(d(i,j)\)

**Step 4.** Put the first arc \((p,q)\) on the list into list A

**Step 5.** Put the next arc on the list into A, except if a circuit can be found with the arcs already in A.

**Step 6.** If all nodes are connected, stop, otherwise go to Step 5.
The problem of modeling, analysis and reconstruction of noisy and/or distorted syntactic patterns is studied. Segmentation errors and primitive extraction errors can be treated as syntax errors and defined in terms of language transformation rules. Three types of error transformations are defined on strings, namely substitution, insertion and deletion. Consequently, the parser constructed according to the grammar generating the strings and the three types of transformations is called the error-correcting parser. This technique is also extended to tree languages. In formulating error-correcting tree automata (ECTA), five types of error-transformations on trees are defined, namely, substitution, split, stretch, branch and deletion. By way of using language transformations, the distance between two sentences can be determined. A definition of distance between a sentence and a language is proposed. Based on this definition, a clustering procedure is proposed, where error-correcting parsers (Cont.)
are employed to determine the distance between an input syntactic pattern and a formed cluster, or a language. Finally, using the error-correcting parsing techniques, real data examples on texture modeling and discrimination are presented.