Recognition of Mathematics Using a Two-Dimensional Trainable Context-free Grammar

by

Jesse F. Hull

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1996

ARCHIVES
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
OCT 15 1996
LIBRARIES

© Jesse F. Hull, MCMXCVI. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part, and to grant others the right to do so.

Author .....

Department of Electrical Engineering and Computer Science

June 11, 1996

Certified by .....

Philip A. Chou

Member of Research Staff, Xerox PARC

Thesis Supervisor

Certified by .....

Mitchell D. Trott

Assistant Professor

Thesis Supervisor

Accepted by .....

Frederic R. Morgenthaler

Chairman, Departmental Committee on Graduate Theses

Recognition of Mathematics Using a Two-Dimensional Trainable Context-free Grammar

by

Jesse F. Hull

Submitted to the Department of Electrical Engineering and Computer Science
on June 21, 1996, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

A method for recognizing both typeset and handwritten mathematics using a two-dimensional trainable context-free grammar is presented. This is done in the context of a communication system view of document recognition by designing a context-free model for the generation and layout of mathematical expressions, then constructing a recognizer based on this model. It is then shown how this model can be trained based on example mathematical expressions.

Thesis Supervisor: Philip A. Chou
Title: Member of Research Staff, Xerox PARC

Thesis Supervisor: Mitchell D. Trott
Title: Assistant Professor

Acknowledgments

First, I must thank my advisor, Phil Chou. He gave me guidance and assistance while also letting me explore my own (sometimes unsuccessful) ideas. He was a pleasure to work with. I would like to say thanks to everyone in the Document Image Decoding (DID) group at Xerox. I also appreciate Mitch Trott, my advisor here at MIT, for helping edit my thesis. To Laura, I will simply say, I wouldn't have survived this past term with out you. Finally, thank you Mom and Dad for giving me the chance to be here and for always believing in me.

Contents

1 Introduction 9

2 Generation and Layout of Mathematical Expressions 12
- 2.1 Introduction . . . 12
- 2.2 Stochastic and Attributed Context-free Grammars . . . 12
- 2.3 Source Model for Mathematical Expressions . . . 14
- 2.4 Model for Encoder . . . 16
  - 2.4.1 Synthesized Attributes . . . 16
  - 2.4.2 Inherited Attributes . . . 17
  - 2.4.3 Synthesis Function . . . 19
  - 2.4.4 Inheritance Function . . . 21
  - 2.4.5 Random Layout . . . 22
- 2.5 Channel Model . . . 24
  - 2.5.1 Handwritten Mathematics . . . 24
  - 2.5.2 Typeset Mathematics . . . 25

3 The Decoder 28
- 3.1 Introduction . . . 28
- 3.2 Derivation of Decoder Equation . . . 28
- 3.3 The Shortest Path Problem . . . 30
  - 3.3.1 The Standard Shortest Path Algorithm . . . 31
  - 3.3.2 The Heuristic A^* Algorithm . . . 32
- 3.4 Using A^* Search for Image Recognition . . . 34
  - 3.4.1 The Heuristic Completion Estimate . . . 36
  - 3.4.2 Other Methods of Improving Decoder Performance . . . 37

4 Performing Alignment 40
- 4.1 Introduction . . . 40
- 4.2 Reasons for Performing alignment . . . 40
- 4.3 Approach to Alignment . . . 41
- 4.4 Automatic Generation of Grammar File . . . 42
- 4.5 Automatic Generation of an Automatic Grammar File Generator . 44

5 Learning of Parameters in Layout Model 47
5.1 Introduction . . . 47
5.2 Maximum Likelihood Estimation of Parameters . . . 47
5.3 Estimation of Grammar Rule Probabilities . . . 49
5.4 Estimation of Channel Parameters . . . 50
5.5 Estimating the Layout Functions . . . 51
5.5.1 Parameterization of the T_{21} Functions . . . 52
5.5.2 Estimation of the T_{21} Functions . . . 54
5.5.3 Estimating the Variance of the Error . . . 57
5.5.4 Parameterization of the F_0 Functions . . . 58
5.5.5 Estimation of the F_0 Function . . . 59
5.5.6 Parameterization of the T_{10} Functions . . . 63
5.5.7 Estimating the T_{10} functions . . . 64

6 Design of the System 66
6.1 Introduction . . . 66
6.2 Interface to Handwritten Mathematics . . . 66
6.3 Stroke Capture . . . 69
6.4 Handwritten Character Recognition . . . 69
6.5 Character Recognition for Typeset Mathematics . . . 70
6.6 Post Processing of Template Matching Data . . . 71
6.7 Extracting Character Positions from a POSTSCRIPT File . . . 72
6.8 Converter . . . 72
6.9 Renderer . . . 72

7 Results 74
7.1 Introduction . . . 74
7.2 Definition of Accuracy . . . 74
7.3 Variation of the Rule Threshold . . . 75
7.4 Effects of the Heuristic Estimate *est* . . . 75
7.5 Effects of the Overlap Percentage . . . 76
7.6 Recognition with a Transcription . . . 77
7.7 Learning Parameters . . . 77
7.7.1 Results of Estimating T_{21} . . . 78
7.7.2 F_0 and T_{10} . . . 81
7.8 Conclusions . . . 82
7.9 Future Directions . . . 82

A The Edit Distance between Two Trees 84

B The Least Squares Solutions to Linear Equations 86

C Equations Used in Experiments 88

List of Figures

1-1 Communication system view 10

2-1 Example parse tree 15
2-2 Message and image views of a document. 16
2-3 Font metric information. 17
2-4 Character concatenation. 18
2-5 Relationships between coordinate systems of parent and children. . 22
2-6 Relationships between coordinate systems of parent and children in the presence of randomness. 23
2-7 Binary asymmetric channel. 25
2-8 Relation between O, O_1, O_2, O_{new}, and O_{inter}. 26
2-9 Black pixels in $X = A - B - C + D$. 27

3-1 Two iterations of the best first algorithm on a sample graph. 32
3-2 An example of A^* search. 34
3-3 The modified A^* algorithm that the decoder uses. 35
3-4 The heuristic completion score estimates $P(O, I(T^\alpha))/P(O, I(T_1^\alpha))$ based on T_1^α. 36

4-1 Example of alignment on "Plus[a,b]". 41
4-2 Block diagram for (a) standard recognition and (b) alignment. . . . 42
4-3 Block diagram of automatic generation of grammar file. 43
4-4 Example of generating a specific grammar file from a parse tree. . . 44
4-5 Block diagram for automatic generation of transcription parser. . . 46

5-1 Overall approach to estimation of parameters. 48
5-2 Example of a nonlinear layout rule. 53
5-3 Relation between various layout functions. 54
5-4 Collecting together instances of multiplication and exponentiation rules. 55
5-5 Four cases to consider when estimating the F_0 function. 60

6-1 Overall block diagram of the equation recognition system. 67
6-2 The interface to input handwritten mathematics. 68
6-3 Example of stroke capture on two characters. 69

6-4 (a) a character model for "x", (b) an example image of an "x", (c) the model overlaying the image, (d) the image overlaying the model, (e) the image and model intersected 71
6-5 Example of converting a parse tree to a variety of formats. 73

7-1 Rule Threshold versus (a) accuracy and (b) total recognition time. . 75
7-2 Rule Threshold versus accuracy (a) with heuristic and (b) without heuristic. Rule threshold versus time (c) with heuristic and (d) without heuristic. 76
7-3 Overlap Percentage versus (a) accuracy and (b) recognition time. . 77
7-4 Examples of an equation laid out using various layout models: (a) original image, (b) initial layout model, (c) layout model after one iteration of training, (d) layout model after two iterations of training. 79
7-5 Examples of an equation laid out using various layout models: (a) original image, (b) initial layout model (including F_0 and T_{10}), (c) layout model after one iteration of training all three layout functions. 81

A-1 Example of the three edit operations performed on ae^x. 85

List of Tables

7.1 The accuracy and recognition time with and without a transcription. 77
7.2 Accuracy while using 20 training examples. 79
7.3 Accuracy while using 50 training examples. 80
7.4 Accuracy while using 5 training examples. 80
7.5 Accuracy while using a varying number of training examples. 82

Chapter 1

Introduction

The goal of this thesis is to explore issues relating to the "recognition" of mathematical notation, where "recognition" means the conversion from a bitmap or set of strokes representing a mathematical expression to a textual representation. Both typeset and handwritten mathematics shall be examined. Some of the question to be explored are:

1. How can mathematical expressions be recognized accurately and efficiently?
2. What is the accuracy/speed trade off involved in recognition?
3. How can one build a recognizer that can be easily extended for new types of mathematical notation?
4. Can one automatically improve or extend a recognizer based on training data?
5. Can customized recognizers that only recognize a subset of mathematical expressions be automatically generated?

A model for the generation of mathematical expressions (and more generally documents) is presented. Then a description of how one can construct a recognizer based on this model is presented. This is done in the context of the communication system view of document recognition. It is then shown how this model can be trained. Finally, some solutions and experiments to address the questions presented above are described.

The communication system view of document recognition described here was first proposed in [11]. In this view, as seen in Figure 1-1, the document, mathematical expression, or message string M is created by the source. This message is

then rendered as an image I by the encoder. The image I then passes through a channel, which typically degrades I in some manner to produce the noisy image O. O is then the input to the decoder that attempts to constructs an estimate $\hat{M}$ of the original message M.

The source assigns a probability to every message string. A message string can be thought of as the information that the author of the document wishes to convey. In general, finding an appropriate probabilistic model for source strings is difficult. Consider, for example, the set of equations that appear in mathematics papers. Presumably, these equations satisfy the property of being mathematically true. Furthermore, the equations that appear most often will be those that mathematicians deem useful or fundamental. Clearly, given an arbitrary equation, deciding whether it is true, not to mention fundamental, is an extremely difficult task. However, in practice, one can provide some information about the character of the set of message strings by modeling them as elements of a regular or context-free language.

The encoder takes a message string and produces a rendered image. When dealing with typeset documents the encoding is typically done automatically by providing programs such as LaTeX or Framemaker with font information as well as the message string. However, even in the typeset case, layout is occasionally done by hand. When recognizing handwritten documents, both the layout and the characters are a matter of individual style.

The channel typically introduces some sort of random noise into the rendered image. The process of printing then scanning is the prototypical example. Examples of noise are randomly flipped bits or randomly skewed text.

With models for the first three elements of the communication channel the notion of an optimal decoder can be defined. The optimal decoder is a deterministic function d^* that maps the observed image O to a message string $\hat{M}$ such that $E(\ell(M, d(O)))$ is minimum, where $\ell(M, \hat{M})$ is a loss function for the particular problem.

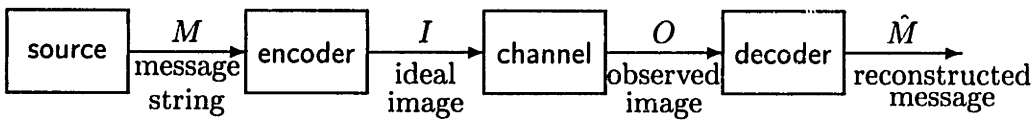


Figure 1-1: Communication system view

Using the communication systems view of document recognition provides a number of benefits. It allows for easy construction of custom recognizers. Additionally, recognition can be improved by modify the source and encoder in response to training data.

Chapter 2

Generation and Layout of Mathematical Expressions

2.1 Introduction

This chapter introduces some concepts needed throughout the thesis. First, definitions and terminology related to grammars and parsing are reviewed. Then the source model for equations, the rendering model for the layout of equations, and the channel noise model are described in detail.

2.2 Stochastic and Attributed Context-free Grammars

In this section, only the concepts necessary for equation recognition are introduced. For a complete treatment of this material, see [1, 7, 8, 9, 10, 2, 13]

A *context-free grammar*, or just *grammar*, is denoted $G = (V, T, P, S)$, where V and T are finite sets of variables and terminals, respectively, and V and T are disjoint sets. P is a finite set of productions; each production is of the form $A \rightarrow \alpha$, where A is a variable and α is a string of symbols from $(V \bigcup T)^*$. Finally, S is a special variable called the start symbol.

A *stochastic context-free grammar* is a context-free grammar where every production rule $r : A \rightarrow \alpha$ has associated with it a real number $P(r)$ such that $0 < P(r) \leq 1$, and for each variable A, $\sum_{r:A\rightarrow\alpha} P(r) = 1$. Furthermore, the

probability of a parse tree T is

$$P(T) = \prod_i P(r_i), \tag{2.1}$$

where r_i is the production rule for the ith internal node of T. This can be written recursively as

$$P(T) = P(r_0) \prod_k P(T_k), \tag{2.2}$$

where $T_1, T_2, \ldots, T_n$ are the children T and r_0 is the production rule for the root of T.

An *attribute grammar* is a context-free grammar in which:

1. each symbol has an associated set of attributes, partitioned into two subsets called the *synthesized* and *inherited* attributes of that grammar symbol, and

2. each production rule has an associated set of functions, partitioned into two subsets called the *synthesis* and *inheritance* functions of that production rule.

The synthesis functions compute the values of the synthesized attributes of the symbol on the left side of the production rule from the values of the synthesized attributes of the symbols on the right side of the production rule. The inheritance functions compute the value of the inherited attributes of the symbols on the right side of the production rule based on the synthesized and inherited attributes of the symbol on the left side of the production rule. Given a parse tree and an attribute grammar with the synthesized attributes supplied for all the terminal nodes in the parse tree, one can then calculate the synthesized attributes for all the nodes in the parse tree in a bottom up fashion. This is done by applying the synthesis function at each internal node in the parse tree. For our purposes the synthesis function will depend only on the synthesized attributes of its children. Similarly, the inherited attributes can be calculated in a top down fashion given the inherited attributes of the root node. It is assumed that the inheritance function depends on only the inherited attributes of its parents and the synthesized attributes of its children. The process of assigning each symbol in a parse tree a set of attributes is called *annotating* or *decorating* the parse tree. Given this framework, annotation can be done with two passes over the parse tree (bottom up followed by top down).

2.3 Source Model for Mathematical Expressions

Much of the framework used to describe the source, encoder, channel, and decoder about to be presented is based on [5] as well as [3]. The goal of the source model is to assign a probability $P(M)$ to every valid message string M. As was stated previously, generating the actual probability that a mathematical expression will appear in a document is intractable. If, however, the set of message strings is described as the synthesized attributes of a stochastic context-free grammar, then the message string represents a syntactically well formed expression. This expression should be a super set of the set of valid mathematical expressions that one would expect to find in documents.

The synthesized attributes (there are no inherited attributes) in this message grammar are simply strings of text. The synthesis functions are functions mapping one or more strings to a new string. The notation

"times[%1, %2]"

denotes a synthesis function that takes two strings as arguments and produces the string obtained by replacing %1 with the first argument and %2 with the second argument. The message string associated with a parse tree is the synthesized string attribute of the root of the parse tree.

For example, the following would be a stochastic context-free grammar producing simple expressions:

$$
\begin{aligned}
\langle Etop \rangle &\rightarrow \langle Etopleft \rangle \langle Eplus \rangle && : 0.75 \quad \text{“\%1\%2]”} \\
\langle Etop \rangle &\rightarrow \langle Eplus \rangle && : 0.25 \quad \text{“\%1”} \\
\langle Etopleft \rangle &\rightarrow \langle Eplus \rangle \text{ ‘=’} && : 1.0 \quad \text{“equal[\%1,”} \\
\langle Eplus \rangle &\rightarrow \langle Eplusleft \rangle \langle T \rangle && : 0.5 \quad \text{“\%1\%2]”} \\
\langle Eplus \rangle &\rightarrow \langle T \rangle && : 0.5 \quad \text{“\%1”} \\
\langle Eplusleft \rangle &\rightarrow \langle Eplus \rangle \text{ ‘+’} && : 1.0 \quad \text{“plus[\%1,”} \\
\langle T \rangle &\rightarrow \text{‘a’} && : 0.3 \quad \text{“a”} \\
\langle T \rangle &\rightarrow \text{‘b’} && : 0.3 \quad \text{“b”} \\
\langle T \rangle &\rightarrow \text{‘c’} && : 0.4 \quad \text{“c”}
\end{aligned}
$$

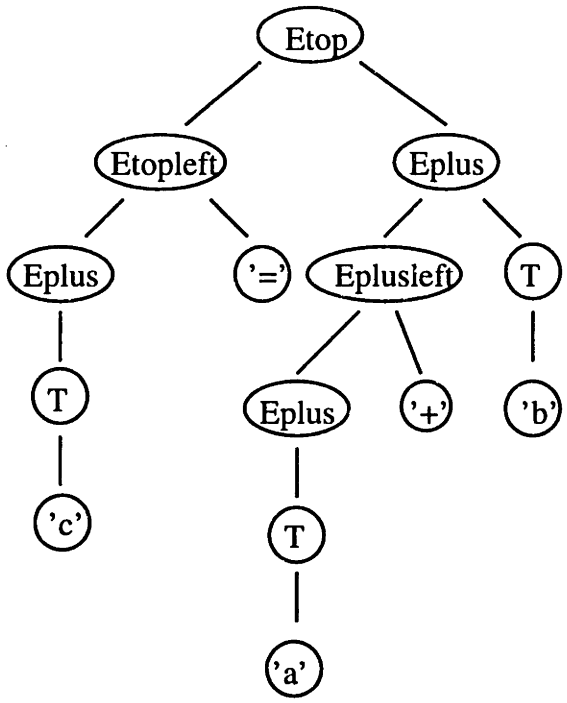


Figure 2-1: Example parse tree

The expression (or message string) "equal[c,plus[a,b]]" is associated with the parse tree shown in Figure 2-1. Furthermore, the probability assigned to this expression is the product of the probabilities of all the rules applied in generating the expression, namely $(0.4)(0.5)(1.0)(0.3)(0.5)(1.0)(0.3)(0.5)(0.75) = .03375$.

Two additional restrictions are made on the source model to simplify the construction of the decoder. First, every production will be of the form $A \rightarrow B$ or $A \rightarrow BC$ where $B, C \in (V \bigcup T)^*$. Grammars satisfying this condition are said to be in *Chomsky normal form.* It can be easily shown that any grammar can be converted to an equivalent one in Chomsky normal form. Secondly, the grammar is assumed to be *unambiguous*, which means every valid message string corresponds to one and only one parse tree.

With these restriction, $P(M)$ can be written using (2.2) as

$$P(M) = P(T) = p(r_0)p(T_1)p(T_2) = p(r)P(M_1)P(M_2), \tag{2.3}$$

where T is parse tree representing M.

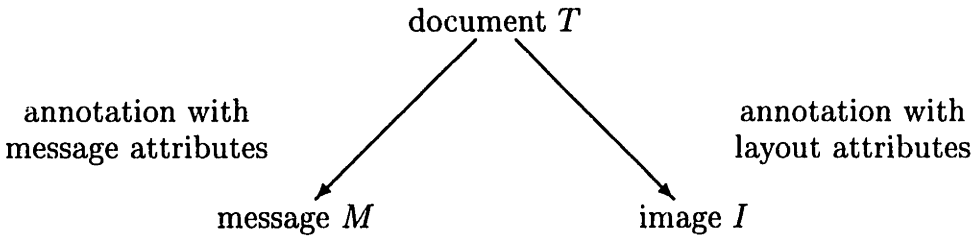


Figure 2-2: Message and image views of a document.

2.4 Model for Encoder

The model for the encoder is based on the same parse tree as the model used for the source. However, the attributes and attribute functions are quite different. The situation is depicted in Figure 2-2. The layout is done by performing two passes over the parse tree. The first pass (bottom up) determines how the children of each node are laid out relative to each other. This essentially is the construction of the synthesized attributes, which will be call *box metrics*. The second pass (top down) determines the position and font size of each character on the page. This is done by constructing a global coordinate transformation for the entire parse tree then propagating this information down the tree. This process is simply the construction of the inherited attributes. A parse tree decorated with both its layout and message attributes will be denoted as T^{α}, in contrast to T, which will represent a parse tree decorated with only its message attributes. The attributes and attribute functions used in this process are now described in detail.

2.4.1 Synthesized Attributes

The synthesized attributes are an extension of the font metric information provided in a single POSTSCRIPT* character. In POSTSCRIPT the font metric information consists of three two dimensional vectors. The lower left corner (ll_x, ll_y), the upper right corner (upr_x, upr_y), and the end point (end_x, end_y). See Figure 2-3. The first two vectors are self explanatory. For characters, the end point is the point at which the origin of the next character will be placed when laying out characters in normal text. See Figure 2-4.

*POSTSCRIPT is a registered trademark of Adobe Systems Inc.

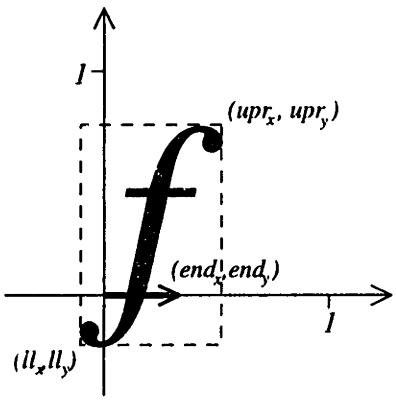


Figure 2-3: Font metric information.

The synthesized attributes for a character (or leaves of the parse tree) are simply the font metrics for that character. The synthesized attributes for non-leaf nodes also consist of three vectors. These more general attributes shall be referred to as *box metrics*, where the first two are again the lower left corner and the upper right corner of the bounding box and the third is some distinguishing point in the frame of the node. The notation $(F_A^{ll_x}, F_A^{ll_y})$, $(F_A^{upr_x}, F_A^{upr_y})$, and $(F_A^{end_x}, F_A^{end_y})$ shall be used to denote the lower left, upper right, and end point of the box metrics for a node A.

2.4.2 Inherited Attributes

To actually layout a mathematical expression one must construct a global coordinated transformation for all of the characters in the image. These transformations specify how the local coordinates of each character are transformed to the global coordinates of the page. In general this transformation can be non-linear. In practice, however, the affine transformation

$$G\colon \begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

is sufficient to represent the type of transformations occurring in equations. And in fact, a uniform scaling translation, which takes the form

$$G\colon \begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

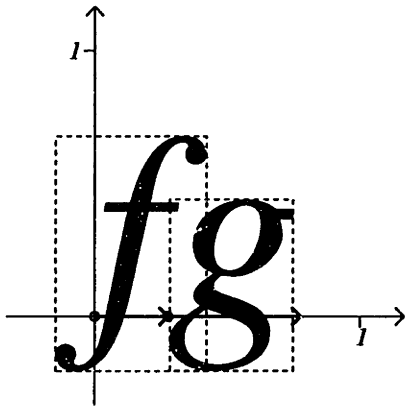


Figure 2-4: Character concatenation.

is assumed. This assumption may be too restrictive for equations because there are characters in mathematics that undergo non-uniform scaling (such as parentheses or square roots). However, for ease of recognizing and specifying layout parameters a uniform scaling translation is used.

A uniform scaling translation G is parameterized by the three values G^{scale}, G^{shift_x}, and G^{shift_y}, which will denote the transformation

$$G\colon \begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} G^{scale} & 0 \\ 0 & G^{scale} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} G^{shift_x} \\ G^{shift_y} \end{bmatrix}$$

Furthermore, if F represents a set of box metrics then $G \circ F$ represents the set of box metrics produced by applying G to every vector in F. Also, if G_1 and G_2 are transformations then $G_1 \circ G_2$ denotes the transformation created by applying G_2 followed by G_1. Finally, G^{-1} denotes the inverse transform given by

$$G^{-1}\colon \begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} \frac{1}{G^{scale}} & 0 \\ 0 & \frac{1}{G^{scale}} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} \frac{-G^{shift_x}}{G^{scale}} \\ \frac{-G^{shift_y}}{G^{scale}} \end{bmatrix}$$

where $G^{scale} \neq 0$.

2.4.3 Synthesis Function

The synthesis function for each rule constructs the box metrics (or equivalently the synthesized attributes) for each internal node in the parse tree. (The box metrics for the leaves are the font metrics for each character.) That is, for an internal node with rule $A \rightarrow BC$ the synthesis function $\mathcal{F}$ maps F_B and F_C to F_A: $F_A = \mathcal{F}(F_B, F_C)$. The synthesis function is composed of three functions T_{21}, T_{10}, and F_0 supplied in the layout grammar. There are two reasons for dividing the synthesis functions into these parts. First, these functions more intuitively match our view of how layout is done. Secondly, they can be composed to form both the synthesis function and the inheritance function. T_{21}, T_{10}, and F_0 are now described.

T_{21} Function

The T_{21} function for a rule of the form $A \rightarrow BC$ is a coordinate transformation from the coordinate system of second node C to the coordinate system of first node B. For a rule of the form $A \rightarrow B$, T_{21} is taken to be the identity since there is no second node. As with the global coordinate transformation, T_{21} is required to be a uniform scaling translation. T_{21} specifies how the equations represented by the subtrees of nodes B and C should be laid out next to one another. T_{21} is a function of the box metrics of B and C only. For example, a T_{21} function for a rule representing the exponentiation of B by C might look like

$$
\begin{aligned}
T_{21}^{scale} &= 0.7 \\
T_{21}^{shift_x} &= F_B^{end_x} \\
T_{21}^{shift_y} &= 0.5(F_B^{upr_y} - F_B^{end_y}),
\end{aligned}
$$

where F_B is box metric for the node B.

These equations specify that the scale of the exponent should be 0.7 of the scale of the expression being exponentiated, the x-coordinate of the origin of the exponent should be the same as if these two expressions were being multiplied (namely equal to the endpoint of B), and the y-coordinate of the origin of the exponent should be halfway between the base line and the top of the expression being exponentiated.

T_{10} function

The T_{10} function for a rule of the form $A \rightarrow BC$ or $A \rightarrow B$ is a coordinate transformation from the coordinate system of B to the coordinate system of A. When the rule is of the form $A \rightarrow B$ this function depends only on the box metrics of B. However, when the rule is of the form $A \rightarrow BC$ this function depends on the box metrics of B and C as well as T_{21}, the transformation between them. Furthermore, the transformation from the coordinate system of C to A is determined by T_{10} and T_{21} and therefore need not be explicitly provided. Continuing with the example of exponentiation the T_{10} function might be

$$\begin{aligned} T_{10}^{scale} &= 1.0 \\ T_{10}^{shift_x} &= 0.0 \\ T_{10}^{shift_y} &= 0.0, \end{aligned}$$

which is simply the identity transformation, as would typically be the case in production rule representing exponentiation or multiplication.

F_0 Function

The F_0 function for a rule of the form $A \rightarrow BC$ or $A \rightarrow B$ is the function that computes the box metrics for the internal node A. When the rule is of the form $A \rightarrow BC$, F_0 takes as arguments the box metrics for B and C *in the coordinate system of* A, which are given by the expressions $\bar{F}_B = T_{10} \circ F_B$ and $\bar{F}_C = T_{10} \circ (T_{21} \circ F_C)$, respectively. Here F_B and F_C are the box metrics for the nodes B and C in their own coordinate system. When the rule is of the form $A \rightarrow B$, F_0 only depends on $\bar{F}_B$. Furthermore, the first two vectors of the box metrics for any node are simply the lower left and upper right coordinates for that node. This means that these two vectors are always calculated in the same manner, namely by

$$\begin{aligned} F_0^{ll_x} &= \min(\bar{F}_B^{ll_x}, \bar{F}_C^{ll_x}) \\ F_0^{ll_y} &= \min(\bar{F}_B^{ll_y}, \bar{F}_C^{ll_y}) \\ F_0^{upr_x} &= \max(\bar{F}_B^{upr_x}, \bar{F}_C^{upr_x}) \\ F_0^{upr_y} &= \max(\bar{F}_B^{upr_y}, \bar{F}_C^{upr_y}). \end{aligned}$$

Therefore, when specifying the F_0 function for each rule in the grammar, one only need supply information about how to calculate the end point. Again, in the case of exponentiation, the F_0 function might look like

$$F_0^{end_x} = \bar{F}_C^{end_x}$$
$$F_0^{end_y} = \bar{F}_B^{end_y}.$$

Definition of Synthesis Function in terms of T_{21}, T_{10}, and F_0

The synthesis function $\mathcal{F}$ can now be defined for each rule $A \rightarrow BC$ in our grammar in terms of the functions T_{21}, T_{10}, and F_0. Let

$$T_{21} = T_{21}(F_B, F_C) \tag{2.4}$$
$$T_{10} = T_{10}(F_B, F_C, T_{21}) \tag{2.5}$$
$$\bar{F}_B = T_{10} \circ F_B \tag{2.6}$$
$$\bar{F}_C = T_{10} \circ (T_{21} \circ F_C). \tag{2.7}$$

Then

$$\mathcal{F}(F_B, F_C) = F_0(\bar{F}_B, \bar{F}_C),$$

where F_B and F_C are the box metrics for the nodes B and C respectively. For rules of the form $A \rightarrow B$ the equations are the same except the terms involving F_C are removed.

2.4.4 Inheritance Function

The inheritance function $\mathcal{G}$ for each rule constructs the global coordinate transformation (or equivalently the inherited attributes) for each non-root node in the parse tree. That is for rules of the form $A \rightarrow B$, $G_B = \mathcal{G}(F_B, G_A)$, and for rules of the form $A \rightarrow BC$, $G_B = \mathcal{G}_B(F_B, F_C, G_A)$, $G_C = \mathcal{G}_C(F_B, F_C, G_A)$. As was stated previously, the inheritance function for rules of the form $A \rightarrow BC$ is composed of the three functions T_{21}, T_{10}, F_0 supplied in the layout grammar. Using (2.4) and

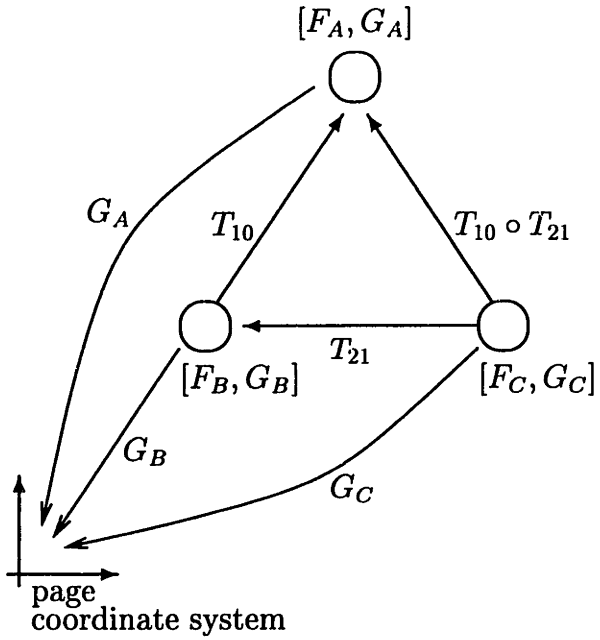


Figure 2-5: Relationships between coordinate systems of parent and children.

(2.5) again $\mathcal{G}$ is defined as

$$G_B = \mathcal{G}_B(F_B, F_C, G_A) \qquad = T_{10}^{-1} \circ G_A \tag{2.8}$$

$$G_C = \mathcal{G}_C(F_B, F_C, G_A) = T_{21}^{-1} \circ (T_{10}^{-1} \circ G_A) \tag{2.9}$$

where $\mathcal{G}_B$ and $\mathcal{G}_C$ are the component functions of $\mathcal{G}$ for nodes B and C, respectively, and G_A is the global transformation for node A. See Figure 2-5. For rules of the form $A \rightarrow B$ the equations are the same except the terms involving F_C are removed and G_C is not calculated.

2.4.5 Random Layout

Although the model presented for the source assigns a probability for each message string, the process of rendering that string into an image is modeled entirely deterministic. An element of randomness is now introduced into the model. There are a number of reasons for doing this. First, it will allow the model to capture a number of different deterministic layout programs. Secondly, if layout is done by hand or if the equation is handwritten, there is probably randomness in the layout process. Finally, by introducing randomness, inaccuracies in the layout model may not be

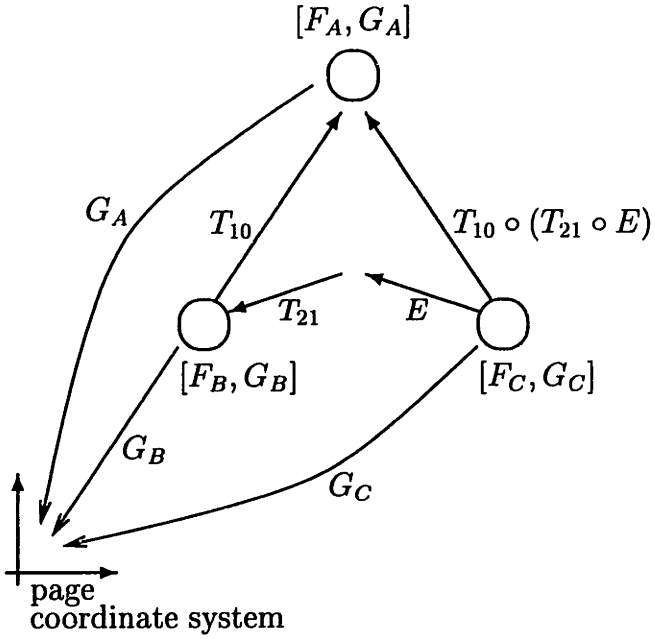


Figure 2-6: Relationships between coordinate systems of parent and children in the presence of randomness.

as costly while performing recognition.

To add randomness to the layout model, only one equation in the previous section is modified: (2.4) is changed to

$$\tilde{T}_{21} = T_{21}(F_B, F_C) \circ E, \tag{2.10}$$

where E is a random transformation. See Figure 2-6. As with previous transformations, E is required to be a uniform scaling translation. Furthermore, it is assumed that $\log(E^{scale})$, E^{shift_x}, E^{shift_y} are all independent Gaussian random variables, where the variance of each is dependent on the particular rule being used and is supplied in the layout grammar. The assumptions that these three variables are independent and Gaussian is not necessary. The effect changing these assumptions would have on recognition is an interesting question not addressed in this thesis.

Intuitively, this modification, before figuring how to layout two nodes relative to each other, first perturbs the second node's box metrics by some random amount. Furthermore, when the layout process was deterministic, every message string produced exactly one attributed layout grammar (assuming an unambiguous grammar) which meant $P(T^{\alpha}|M)$ was a degenerative probability distribution (0 except when

$T^\alpha = M(T^\alpha)))$. However, when layout is modeled with randomness,

$$P(T^\alpha|M) = \prod_i P_E(r_i), \tag{2.11}$$

where $P_E(r_i)$ is the probability of E, the error transformation used in rule r_i. Using (2.2) this can be written recursively as

$$P(T^\alpha|M) = P_E(r_i)P(T_1^\alpha|M_1)P(T_2^\alpha|M_2), \tag{2.12}$$

where T_1^α and T_2^α are the left and right subtrees of T^α and M_1 and M_2 are the corresponding message strings. Furthermore, by Bayes' rule

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

Since M is a deterministic function of T^α this becomes

$$P(T^\alpha) = P(M)P(T^\alpha|M).$$

Substituting (2.3) and (2.12) into this gives

$$\begin{aligned} P(T^\alpha) &= p(r_i)P(M_1)P(M_2)P_E(r_i)P(T_1^\alpha|M_1)P(T_2^\alpha|M_2) \\ &= p(r_i)P_E(r_i)P(T_1^\alpha)P(T_2^\alpha). \end{aligned} \tag{2.13}$$

2.5 Channel Model

2.5.1 Handwritten Mathematics

The model for the channel varies depending on whether the mathematics is handwritten or typeset. With handwritten mathematics it is assumed that the image is not degraded at all in the channel. Typically, the input is received from some sort of pen device. This means there may be noise in the way the characters are drawn or in there position relative to each other, but these two kinds of noise should be captured by the lexical analysis and the random layout. One could attempt to model extraneous strokes using the channel. For simplicity though, all strokes are assumed to be part of the mathematics being drawn. The remaining work assumes typeset mathematics. When dealing with the handwritten case, the equations that follow

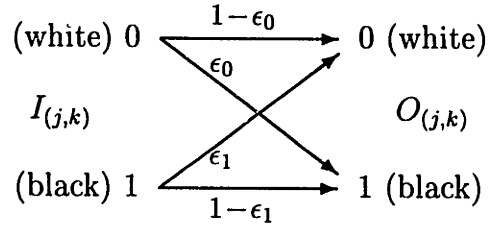


Figure 2-7: Binary asymmetric channel.

remain the same except that the terms involving the channel should be ignored.

2.5.2 Typeset Mathematics

When dealing with typeset mathematics, the channel is modeled as independent binary asymmetric noise $P(O|I)$. Every white bit in the ideal image I flips with probability ϵ_0 and every black bit with probability ϵ_1. See Figure 2-7. Thus, if the ideal image pixel $I^{(j,k)}$ equals 0 (white) then the probability that $O^{(j,k)}$ will equal $x \in \{0,1\}$ is $\epsilon_0^x \bar{\epsilon}_0^{(1-x)}$, where $\bar{\epsilon}_0 = 1 - \epsilon_0$. Similarly, if the ideal image pixel $I^{(j,k)}$ equals 1 (black) then the probability that $O^{(j,k)}$ will equal x is $\epsilon_1^{(1-x)} \bar{\epsilon}_1^x$. Therefore,

$$P(O|I(T^\alpha)) = \prod_{I^{(j,k)}=0} \epsilon_0^{O^{(j,k)}} \bar{\epsilon}_0^{(1-O^{(j,k)})} \prod_{I^{(j,k)}=1} \epsilon_1^{1-O^{(j,k)}} \bar{\epsilon}_1^{(O^{(j,k)})}. \tag{2.14}$$

As before, $P(O|I(T^\alpha))$ shall be expressed recursively in terms of $P(O_1|I_1(T_1^\alpha))$ and $P(O_2|I_2(T_1^\alpha))$. Here $I_1(T_1^\alpha)$ and $I_2(T_2^\alpha)$ are the subimages of the ideal image generated by the subtrees T_1^α and T_2^α, respectively, and O_1 and O_2 are the subimages of the observed image defined to have the same bounding box as I_1 and I_2, respectively. Assuming that all points in the ideal image I are accounted for by the lexical analysis, then all other black points in O_{new} are noise. The calculation of $P(O|I(T^\alpha))$ thus depends only on the number of white and black points in O_{new} and O_{inter}. Hence,

$$P(O|I(T^\alpha)) = \frac{\left(\prod_{(j,k):O_{new}^{(j,k)}} \epsilon_0^{O_{new}^{(j,k)}} \bar{\epsilon}_0^{(1-O_{new}^{(j,k)})} \right) \left(P(O_1|I_1(T_1^\alpha)) P(O_2|I_2(T_2^\alpha)) \right)}{\left(\prod_{(j,k):O_{inter}^{(j,k)}} \epsilon_0^{O_{inter}^{(j,k)}} \bar{\epsilon}_0^{(1-O_{inter}^{(j,k)})} \right)}, \tag{2.15}$$

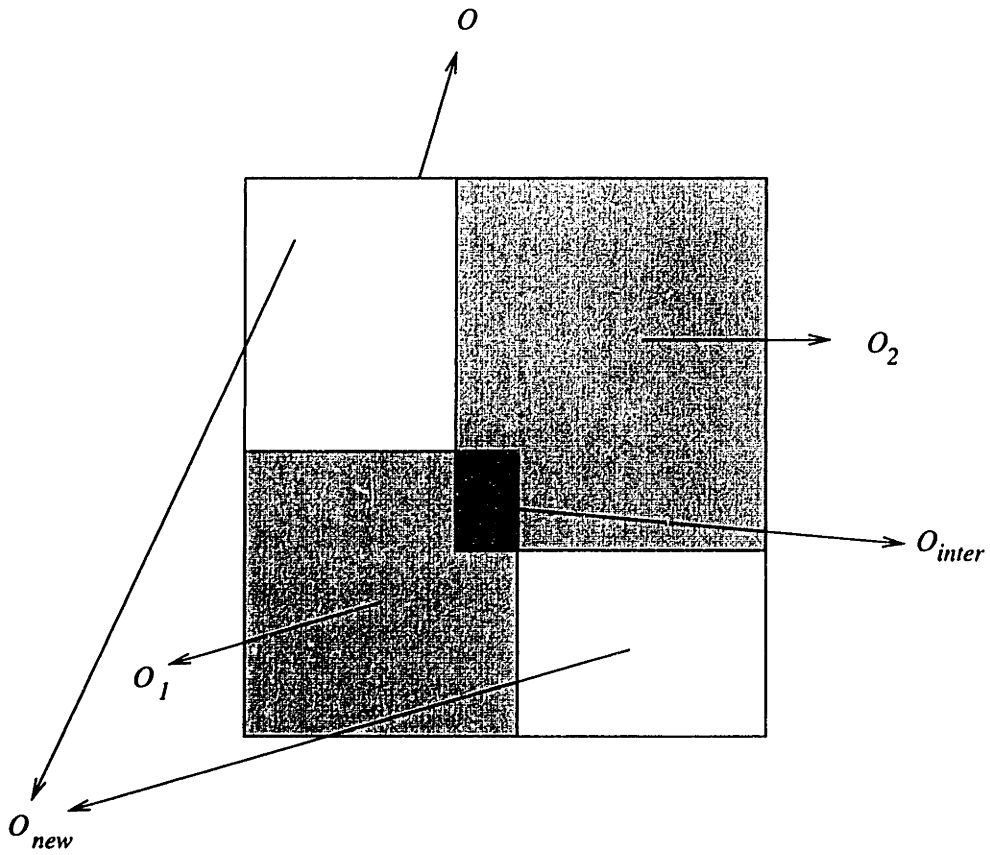


Figure 2-8: Relation between O, O_1, O_2, O_{new}, and O_{inter}.

where it is assumed that $I_1 \bigcap I_2 = \emptyset$ (i.e., the black points of I_1 and I_2 do not intersect.) Equation (2.15) is equivalent to

$$P(O|I(T^\alpha)) = \frac{\epsilon_0^{\text{black points in } O_{new}} \bar{\epsilon}_0^{\text{white points in } O_{new}} P(O_1|I_1(T_1^\alpha))P(O_2|I_2(T_2^\alpha))}{\epsilon_0^{\text{black points in } O_{inter}} \bar{\epsilon}_0^{\text{white points in } O_{inter}}}.$$

Now, letting

$$\begin{aligned} \text{black}(O) &= \text{number of black points in } O, \\ \text{white}(O) &= \text{number of white points in } O, \\ \text{area}(O) &= \text{number of points in } O, \end{aligned}$$

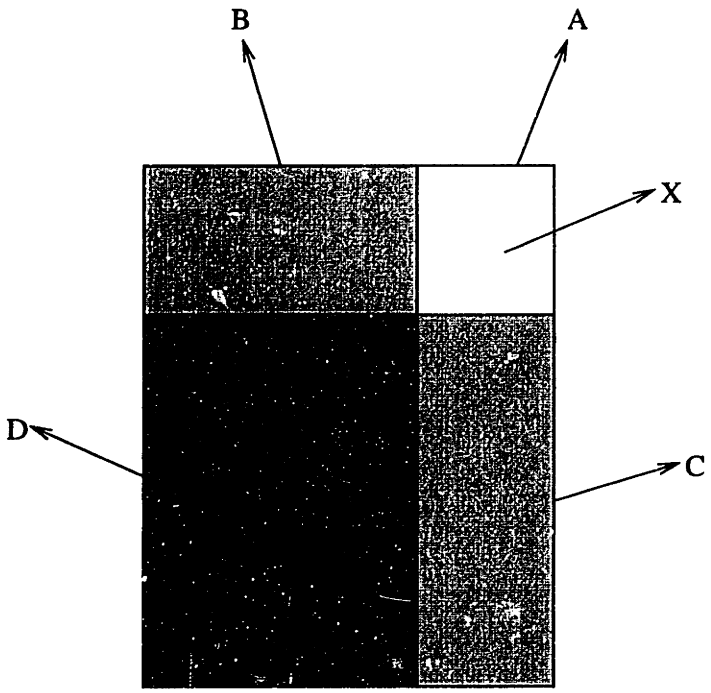


Figure 2-9: Black pixels in $X = A - B - C + D$.

the number of black and white points in O_{new} can be calculated as

$$\begin{aligned}\text{black}(O_{new}) &= \text{black}(O) - \text{black}(O_1) - \text{black}(O_2) + \text{black}(O_1 \cap O_2)\\ \text{white}(O_{new}) &= \text{area}(O_{new}) - \text{black}(O_{new}).\end{aligned}$$

The number of black points in any rectangular region can easily be calculated by maintaining an array $A^{(i,j)}$ of the number of black points in the rectangular region defined by the origin of the page and the point (i, j). See Figure 2-9. Therefore, letting

$$p_C(O) = \frac{\epsilon_0^{\text{black}(O_{new})}\bar{\epsilon}_0^{\text{white}(O_{new})}}{\epsilon_0^{\text{black}(O_{inter})}\bar{\epsilon}_0^{\text{white}(O_{inter})}}, \tag{2.16}$$

gives

$$P(O|I(T^\alpha)) = p_C(O)P(O|I_1(T_1^\alpha))P(O|I_2(T_2^\alpha)), \tag{2.17}$$

a recursive equation for $P(O|I)$.

Chapter 3

The Decoder

3.1 Introduction

In this chapter a recursive equation for the optimal message string is derived. This equation forms the basis for the dynamic programming algorithm used to implement the decoder. After describing the decoder in detail, a few methods of improving its performance are explained.

3.2 Derivation of Decoder Equation

Recall that the optimal decoder d^* minimizes $E(\ell(M, d^*(O)))$, where $\ell(M, \hat{M})$ is some loss function. Choosing

$$\ell(M, \hat{M}) = \begin{cases} 0 & \text{if } M = \hat{M} \\ 1 & \text{otherwise} \end{cases}$$

for the loss function, it is a well known result that the optimal decoder is given by

$$d^*(O) = \arg\max_M P(M|O).$$

By Bayes' rule,

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

Furthermore, because $M \rightarrow I \rightarrow O$ is a Markov chain,

$$P(M,O) = \sum_I P(M)P(I|M)P(O|I). \tag{3.1}$$

Hence,

$$d^*(O) = \arg\max_M \sum_I P(M)P(I|M)P(O|I). \tag{3.2}$$

Further noting that $M \rightarrow T^\alpha \rightarrow I$ is a Markov chain gives

$$P(M)P(I|M) = P(M,I) = \sum_{T^\alpha} P(T^\alpha)P(M|T^\alpha)P(I|T^\alpha),$$

and, since M can be deterministically calculated from T^α,

$$P(M,I) = \sum_{T^\alpha : M(T^\alpha)=M} P(T^\alpha)P(I|T^\alpha).$$

Rewriting $\sum_I P(M)P(I|M)P(O|I)$ as

$$\begin{aligned}\sum_I P(M)P(I|M)P(O|I) &= \sum_I \sum_{T^\alpha : M(T^\alpha)=M} P(T^\alpha)P(I|T^\alpha)P(O|I) \\ &= \sum_{T^\alpha : M(T^\alpha)=M} P(T^\alpha) \sum_I P(I|T^\alpha)P(O|I) \\ &= \sum_{T^\alpha : M(T^\alpha)=M} P(T^\alpha)P(O|T^\alpha),\end{aligned}$$

gives for the optimal decoder

$$d^*(O) = \arg\max_M \sum_{T^\alpha : M(T^\alpha)=M} P(T^\alpha)P(O|T^\alpha). \tag{3.3}$$

Equation (3.3) is difficult to implement in practice because it involves a summation over all trees T^α such that $M(T^\alpha) = M$. However, the fact that

$$\sum_{T^\alpha : M(T^\alpha)=M} P(T^\alpha)P(O|T^\alpha) \geq \max_{T^\alpha : M(T^\alpha)=M} P(T^\alpha)P(O|T^\alpha)$$

can be used to define a suboptimal decoder

$$\begin{aligned} d(O) &= \arg\max_{M} \max_{T^\alpha : M(T^\alpha)=M} P(T^\alpha)P(O|T^\alpha) \\ &= M\left(\arg\max_{T^\alpha} P(T^\alpha)P(O|T^\alpha)\right). \end{aligned}$$

This decoder is optimal if there is only one T^α such that $M(T^\alpha) = M$. Otherwise, it is only approximately optimal, with the approximation most accurate when there are few other T^α with high probability such that $M(T^\alpha) = M$. This is usually the case in practice. $\hat{T}^\alpha$ is defined to be the optimal attributed parse tree given by

$$\begin{aligned} \hat{T}^\alpha &= \arg\max_{T^\alpha} P(T^\alpha)P(O|T^\alpha) \\ &= \arg\max_{T^\alpha} P(O, T^\alpha) \\ &= \arg\max_{T^\alpha} P(O, I(T^\alpha)) \end{aligned}$$

It is sufficient to find $\hat{T}^\alpha$ since $\hat{M}$ can be reconstructed from $\hat{T}^\alpha$. Using (2.13) and (2.17) gives

$$P(O, I(T^\alpha)) = \begin{cases} P(O, I(c)) & \text{if } T^\alpha \text{ is a terminal node representing a character } c. \\ p(r_i)p_E(r_i)p_C(O) & \text{otherwise.} \\ P(O_1, I_1(T_1^\alpha))P(O_2, I_2(T_2^\alpha)) & \end{cases} \tag{3.4}$$

This equation can be used to construct a dynamic programming algorithm to find $\hat{T}^\alpha$. This algorithm is a modification of the standard A^* search used for finding the shortest path between a goal node and a start node in a directed graph. First, the standard A^* search procedure is described. Then it is generalized to construct parse trees for an image. For a complete treatment of the A^* search algorithm see [15, 20].

3.3 The Shortest Path Problem

A *graph* G consists of a set of *nodes* N and a set of *directed edges* E connecting pairs of nodes. An edge e going from node n to n' shall be denoted (n, n'). Every

graph that is considered shall have two distinguished nodes: the *start node* s and the *target node* t. If there is an edge from n to n' then n' is said to be a *successor* of n and n is said to be the *parent* of n'. Every edge $e \in E$ has associated with it a real number $d(e)$. This number is referred to as the *length* of edge e. A *path* from node n_1 to node n_k is a sequence of nodes $n_1, n_2, n_3, \ldots, n_k$ such that n_{i+1} is a successor of n_i for every $1 \leq i < k$. The length of a path is the sum of the lengths of the edges in the path. The *shortest path* from n to n' is the path from n to n' that has minimum length. The shortest path problem consists of finding the shortest path from the start node s to the target node t.

3.3.1 The Standard Shortest Path Algorithm

The standard shortest path algorithm is called *best first*. This algorithm keeps two lists of nodes. The first is called OPEN and consists of nodes that have been reached but whose successors have not been explored. The second, called CLOSED, consists of all nodes that have been reached and whose successors have been explored. The algorithm terminates when the target node t is removed from the open list or when the open list is empty. For an example of the algorithm in progress, see Figure 3-1. Formally, the algorithm is as follows:

The Best First Algorithm

1. Put the start node s on a list OPEN of unexpanded nodes.
2. If OPEN is empty, exit with failure; no solutions exist.
3. Remove from OPEN a node n for which $d(s, n)$ is minimum (breaking ties arbitrarily), and place it on a list called CLOSED to be used for expanded nodes.
4. Expand node n, generating all its successors with pointers back to n.
5. If any of n''s successors are goal nodes, exit successfully with the solution obtained by tracing the path along the pointer from the goal back to s.
6. For every successor n' of n:
   (a) Calculate $d(s, n')$.

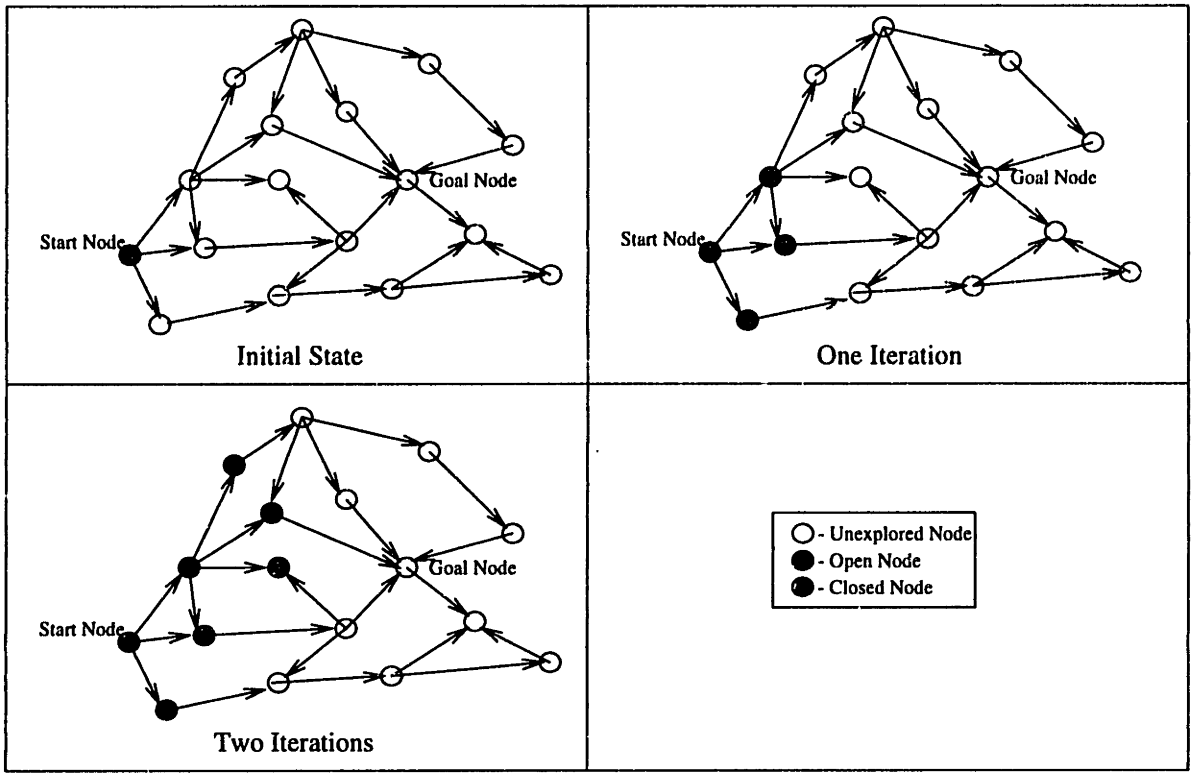


Figure 3-1: Two iterations of the best first algorithm on a sample graph.

(b) If n' was neither on OPEN, nor on CLOSED, add it to OPEN. Attach a pointer from n' back to n. Assign newly computed $d(s, n')$ to node n'.

(c) If n' already resided on OPEN or CLOSED, compare the newly computed $d(s, n')$ with the value previously assigned to n'. If the old value is lower, discard the newly generated node. If the new value is lower substitute it for the old. If the matching node n' resided on CLOSED, move it back to OPEN.

7. Go to step 2.

3.3.2 The Heuristic A^* Algorithm

The A^* algorithm is similar to the best first algorithm. The only change is the manner in which the OPEN list is sorted. In the best first algorithm the node n for which $d(s, n)$ is minimum is removed from the closed list. This corresponds to removing the node that is closest to the start node. However, what would be most desirable would be to remove the node that is most likely to be part of the shortest

path. To capture this idea a new function $est(n, n')$, which is an estimate of the distance between n and n', is created. Using this estimate OPEN is sorted based on the quantity

$$f(n) = d(s, n) + est(n, t).$$

It can be shown that if $est(n, n') \leq d(n, n')$ for all n and n' then the A^* algorithm always finds the shortest path.

The A^* Algorithm

1. Put the start node s on OPEN.
2. If OPEN is empty, exit with failure.
3. Remove from OPEN and place on CLOSED a node n for which $f(n)$ is minimum.
4. If n is a goal node, exit successfully with the solution obtained by tracing back the pointers from n to s.
5. Otherwise, expand n, generating all its successors, and attach to them the pointers back to n. For every successor n' of n:
   (a) If n' is not already on OPEN or CLOSED, calculate $f(n')$ and add n' to OPEN.
   (b) If n' is already on OPEN or CLOSED, direct its pointers along the path yielding the lowest $d(s, n')$.
   (c) If n' required pointer adjustment and was found on CLOSED, reopen it.
6. Go to step 2.

The prototypical example of A^* search is finding the shortest path on a planar graph where the distance between two nodes is just their Cartesian distance. A lower bound for the distance between n and the target node t is the Cartesian distance between them. The shortest path from n to t going through points on

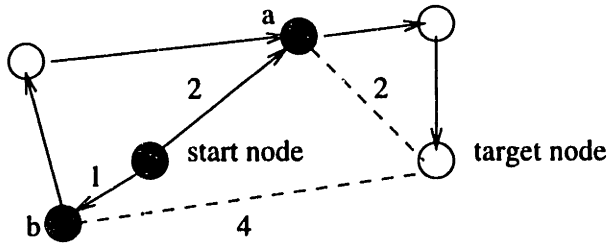


Figure 3-2: An example of A^* search.

the graph must be as least as long as the straight line distance from n to t. For example, in Figure 3-2,

$$f(a) = d(s,a) + est(a,t) = 2 + 2 = 4 \tag{3.5}$$

$$f(b) = d(s,b) + est(b,t) = 1 + 4 = 5 \tag{3.6}$$

hence node a will be expanded first. This is a case where the additional information provided by the heuristic helped properly guide the search. If the heuristic had not been used node b would have been expanded first, since $1 = d(s,b) < d(s,a) = 2$.

3.4 Using A^* Search for Image Recognition

The A^* algorithm needs only slight modification to perform image recognition. The algorithm is depicted in Figure 3-3. Formally:

1. Put the all terminal nodes (the results of the lexical analysis) on OPEN.
2. If OPEN empty, exit with failure.
3. Remove from OPEN and place on CLOSED a T^α for which $f(T^\alpha)$ is minimum.
4. If T^α is the start symbol and describes the whole image, exit successfully with the solution obtained by constructing M from T^α.
5. Otherwise construct all nodes T^α_{new} that can be generated from T^α and a node on CLOSED using a rule in the grammar. For every new generated node T^α_{new}
   (a) If T^α_{new} is not already on OPEN or CLOSED, calculate $f(T^\alpha_{new})$ and add T^α_{new} to OPEN.
   (b) If T^α_{new} is identical to a node T^α_{old} already on OPEN or CLOSED, then

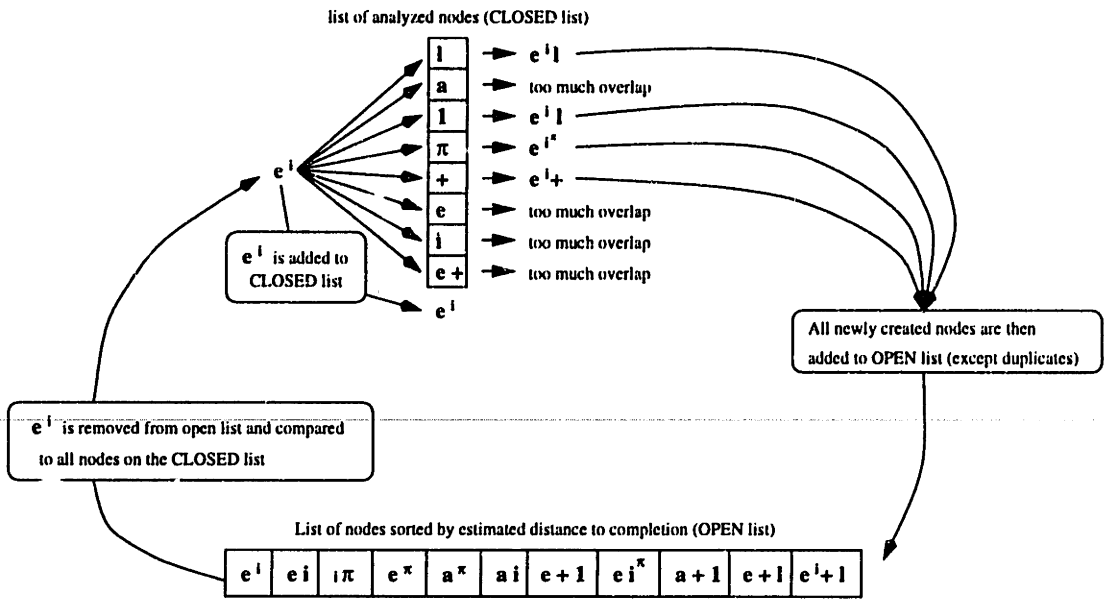


Figure 3-3: The modified A^* algorithm that the decoder uses.

i. if $d(T^{\alpha}_{old})$ is less than $d(T^{\alpha}_{new})$ discard T^{α}_{new}.

ii. if $d(T^{\alpha}_{new})$ is less than $d(T^{\alpha}_{old})$ discard T^{α}_{old} and add T^{α}_{new} to OPEN.

6. Go to step 2.

In this algorithm,

$$d(T^{\alpha}) = -\log(P(O, I(T^{\alpha}))) \tag{3.7}$$

$$f(T^{\alpha}) = d(T^{\alpha}) + est(T^{\alpha}), \tag{3.8}$$

where $est(T^{\alpha})$ is the heuristic completion estimate which we shall describe in Section 3.4.1.

In step 1 $d(T^{\alpha})$ (and therefore $f(T^{\alpha})$) can be calculated for all the terminal nodes because by (3.4) and (3.7)

$$d(T^{\alpha}) = -\log(P(O, I(c))),$$

where c is the character that T^{α} represents.

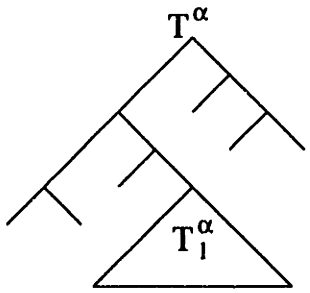


Figure 3-4: The heuristic completion score estimates $P(O, I(T^\alpha))/P(O, I(T_1^\alpha))$ based on T_1^α.

And in step 5 $d(T_{new}^\alpha)$ can be calculated because again, by (3.4) and (3.7)

$$d(T_{new}^\alpha) = d(T^\alpha) + d(T_{closed}^\alpha) - \log(p(r_i)p_E(r_i)p_C(O)),$$

where T_{closed}^α is the node from CLOSED used to create T_{new}^α.

3.4.1 The Heuristic Completion Estimate

Recall that the heuristic completion estimate must be an optimistic estimate of distance to completion. In our context, distance is the negative log likelihood of an attributed parse tree T^α. Therefore, if T^α is a completed parse tree and T_1^α a subtree (as in Figure 3-4) then the constraint that *est* must be a optimistic estimate of distance to completion means that

$$est(T_1^\alpha) \leq -\log \frac{P(O, I(T^\alpha))}{P(O, I_1(T_1^\alpha))}. \tag{3.9}$$

Therefore, a valid (yet not particularly useful) estimate would be $est(T_1^\alpha) = 0$ for all T_1^α. In general, when constructing an *est* function one should be able to estimate $P(O, I(T^\alpha))$ based on T_1^α.

Let O_1 be the portion of O defined to have the same bounding box as $I_1(T_1^\alpha)$ also let $\bar{I}_1 = I - I_1$ and $\bar{O}_1 = O - O_1$. If all of $\bar{O}_1$ corresponds exactly to our layout model and there are no stray bits then

$$P(O, I(T^\alpha)) = \left(\prod_i p(r_i)p_E(r_i)\right) P(\bar{O}_1|\bar{I}_1 = \bar{O}_1)P(O, I_1(T_1^\alpha)),$$

where the r_i's are the rules used in T^α but not in T_1^α, and $P(\bar{O}_1|\bar{I}_1 = \bar{O}_1)$ is proba-

bility that the observed image is $\bar{O}_1$ given that it is predicted to be $\bar{O}_1$. Therefore, for any T^α,

$$P(O, I(T^\alpha)) \leq P(\bar{O}_1|\bar{I}_1 = \bar{O}_1)P(O, I(T_1^\alpha)).$$

Hence by (3.9) a valid estimator is

$$est(T_I^\alpha) = -\log(P(\bar{O}_I|\bar{I}_I = \bar{O}_I)).$$

Using (2.14) gives

$$est(T_I^\alpha) = -\log\left((1-\epsilon_0)^{\text{white}(\bar{O}_1)}(1-\epsilon_1)^{\text{black}(\bar{O}_1)}\right).$$

See Section 7.4 for results on how this estimator performs.

3.4.2 Other Methods of Improving Decoder Performance

Bounding $p_E(r_i)$

Recall equation (3.4):

$$\begin{aligned} P(O, I(T^\alpha)) =& p(r_i)p_E(r_i)p_C(O) \\ & P(O_1, I_1(T_1^\alpha))P(O_2|I_2(T_2^\alpha)). \end{aligned}$$

The $p_E(r_i)$ term describes how well the trees T_1^α and T_2^α are laid out relative to to one another. For example, let r_i represent a rule for multiplication. As $p_E(r_i)$ decreases it becomes more and more unlikely that T_1^α and T_2^α represent two expressions being multiplied. However, because $p_E(r_i)$ is a Gaussian distribution it never drops to zero. This means there is a non-zero probability that any two nodes in our grammar can be combined. Furthermore, if T^α is a large parse tree representing a significant portion of the image and T_1^α is a small parse tree composed of two very unlikely subtrees then it may be the case that $P(O|T^\alpha)$ is less then $P(O|T_1^\alpha)$. This means that the decoder removes T_1^α from OPEN before T^α. However, the decoder should not waste time processing T_1^α because it has such a small probability of being in the final parse tree. Therefore, to capture the notion that small unlikely expressions should be ignored, the decoder discards any node T^α in which $p_E(r_i) < thres$. The quantity *thres* shall be referred to as the rule threshold and should be a value

appropriately chosen given the accuracy of the layout model and the quality of the image. Section 7.3 explores how the value of *thres* affects the speed and accuracy of recognition.

Limiting Overlap

As before, let $I_1(T_1^\alpha)$ and $I_2(T_2^\alpha)$ be the subimages of the ideal image generated by the subtrees T_1^α and T_2^α, respectively, and O_1 and O_2 be the subimages of the observed image defined to have the same bounding box as I_1 and I_2, respectively. In Section 2.5.2 the subimages O_1 and O_2 are allowed to have arbitrary overlap. This means that the layout model allows two terms in an equation to be placed on top of each other. Although the bounding boxes of adjacent characters may overlap somewhat, the probability of them being directly on top of one another should be quite small. To capture this intuition the decoder has an *overlap* parameter. The decoder discards a node if

$$\frac{\text{black}(O_1 \cap O_2)}{\text{black}(O_1)} > overlap$$

or

$$\frac{\text{black}(O_1 \cap O_2)}{\text{black}(O_2)} > overlap.$$

See Section 7.5 for a discussion on how the value of this parameter effects recognition.

The Definition of Identical

Recall that step 5b of the decoder algorithm is
5(b) If T_{new}^α is identical to a node T_{old}^α already on OPEN or CLOSED, then ...
The decoder could define two trees to be identical when they have the same structure and identical attributes. This definition would serve little purpose for optimizing the decoder's algorithm because it can be shown that the decoder will never generate two trees which satisfy this restrictive definition of identical. However, because our layout rules only depend on the box metrics of the root nodes, two trees whose root nodes have identical box metrics will be indistinguishable to the decoder. Therefore, two trees are defined to be identical when the box metrics of there root nodes are the

same. For example, when recognizing an image of $2^n xy$ the decoder will probably generate the parse trees "Times[Power[2,n],x]" and "Times[Times[2, n], x]". If the layout model is accurate then then first tree should have higher probability. Furthermore, the box metrics for the roots of both parse trees should be identical. Therefore, the second parse tree should be discarded. This speeds performance because the decoder no longer generates the parse tree "Times[Times[Times[2, n], x], y]".

Chapter 4

Performing Alignment

4.1 Introduction

A transcription is a textual representation of the parse tree for the image. For example, in Figure 4-1 the string "Plus[a, b]" is the transcription for the image. However, even if one has a transcription there is still the problem finding a correspondence between the transcription and the image, or equivalently, determining where each of the characters in the transcription are on the image. This process is called *aligning* the transcription with the image or just *alignment.*

4.2 Reasons for Performing alignment

There are two reasons to perform alignment. The first, and most important for this work, is that it allows for the measurement of actual values for various parameters in our model. (See Chapter 5.) The second reason is that performing alignment allows one to edit images. Suppose, for example, that a user wished to change the "a" in Figure 4-1 to an "x". This requires knowing which part of the image represents the character "a". With this information the relevant part of the image can be replaced with the bitmap for the character "x". See [4] for a general discussion of image editing. The current system does not support image editing. However, in the current system, all the basic functionality to implement image editing is available.

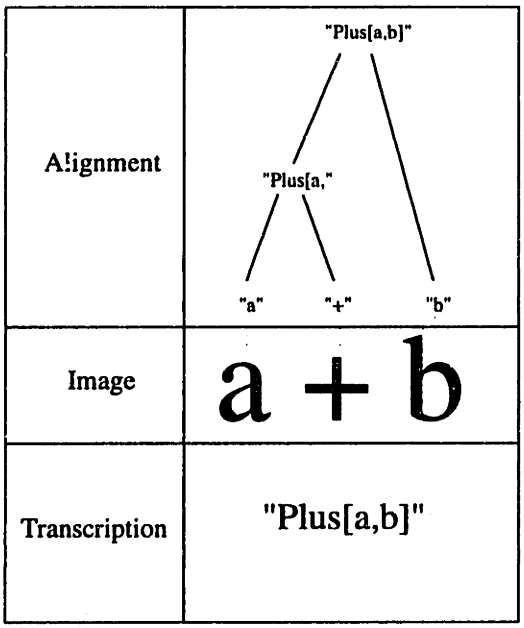


Figure 4-1: Example of alignment on "Plus[a,b]".

4.3 Approach to Alignment

The approach to alignment is straightforward. The equation parser (or decoder) normally performs recognition as depicted in Figure 4-2(a). Because the grammar file is not built into the decoder but taken in as input it can be easily modified. This means, in particular, a grammar that will only parse one particular equation can be used as input into the decoder. (See Figure 4-2(b).) For example, the following grammar file can only generate "Plus[a, b]"

$$\begin{aligned} \langle S \rangle &\rightarrow \langle Eplus \rangle & &: 1.0 \text{ "\%1"} \\ \langle Eplus \rangle &\rightarrow \langle Eplusleft \rangle \text{ 'b'} & &: 1.0 \text{ "\%1\%2]"} \\ \langle Eplusleft \rangle &\rightarrow \text{'a''+'} & &: 1.0 \text{"plus[\%1,"} \end{aligned} \tag{4.1}$$

Note that every expression has exactly one production rule, whose probability is 1.0, due to the grammar file generating only one mathematical expression.

If the decoder uses this grammar file to recognize an image the only answer it will produce is "Plus[a, b]". Furthermore, when the decoder performs recognition with this grammar the number of cases it must consider is substantially reduced. For example, when using a general grammar the decoder must consider the possibility

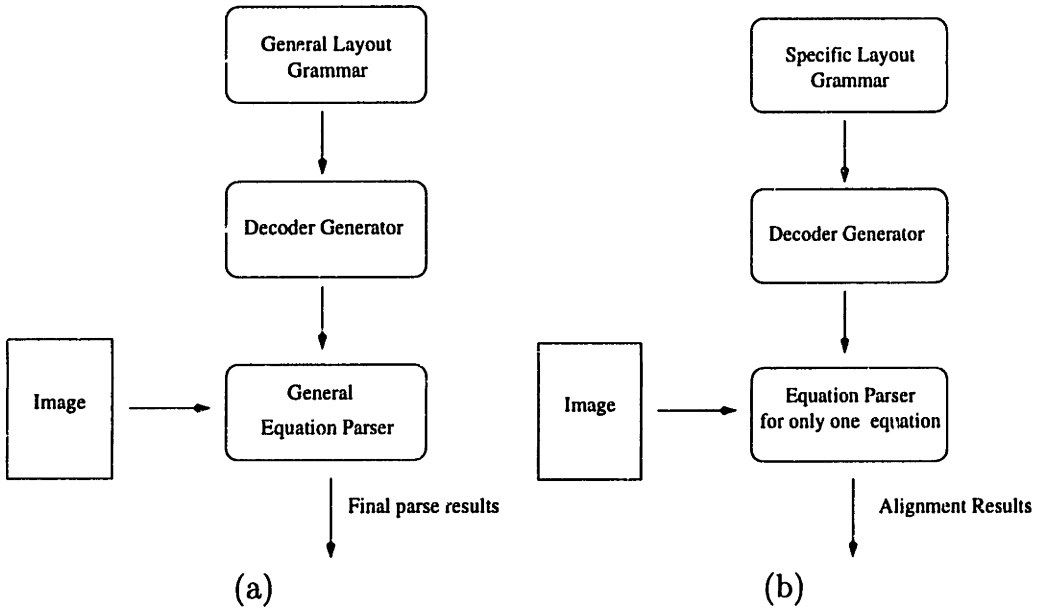


Figure 4-2: Block diagram for (a) standard recognition and (b) alignment.

that the image in Figure 4-1 is "Times[a, b]". (This would mean the the "+" was all noise.) However, if the decoder recognizes the image using the specific grammar above it never considers the possibility of "Times[a,b]" because the grammar file does not even contain this construct.

The fact that the decoder needs to examine considerably fewer cases when performing recognition with a specific grammar file has two consequences. First, the threshold for discarding newly created nodes (see Section 3.4.2) can be set very high (i.e., few nodes are discarded). Secondly, since this threshold is much higher, the model for the way in which equations are laid out does not have to be as accurate to get positive results quickly.

Thus, the decoder can perform alignment much more quickly and with much greater accuracy than when performing real recognition (see Section 7.6) even with a poor model for layout. This certainly is not surprising, because performing alignment is a substantially easier task then performing general recognition.

4.4 Automatic Generation of Grammar File

The basic approach described above certainly works. However, if there are a

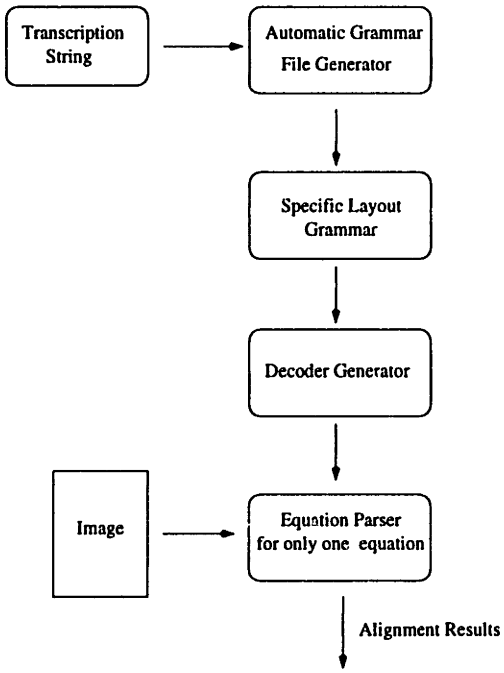


Figure 4-3: Block diagram of automatic generation of grammar file.

large number of equations, constructing a custom grammar file by hand for each one is tedious and error prone. Therefore, an automatic grammar file generator (see Figure 4-3) was implemented. Given a transcription string such as "Plus[a,b]" the generator produces the grammar file in (4.1).

The construction of a grammar file from the transcription string proceeds in two steps. First, the grammar file generator parses the transcription string. To do this it is necessary to describe the set of all transcription strings using a context-free grammar. This somewhat limits the set of transcription strings that can be parsed, but in practice, mathematics is quite naturally represented as the strings of a context-free grammar. Second, the grammar file generator produces the custom grammar file from the parsed transcription string by outputting one or two rules for each node in the parse tree. Sometimes two rules are needed because the custom grammar file must be in Chomsky normal form. For example, Figure 4-4 shows how to construct the custom grammar file for the string "Plus[Times[a, b], c]".

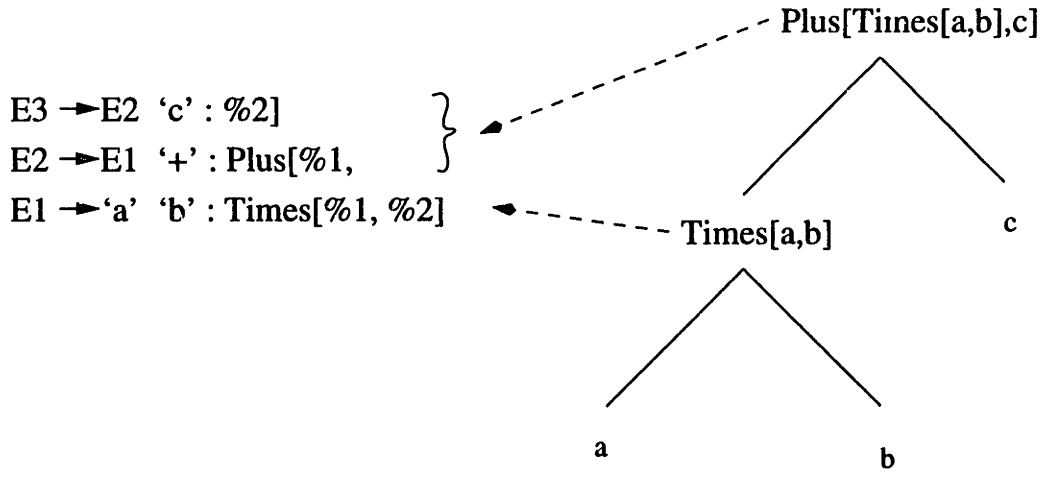


Figure 4-4: Example of generating a specific grammar file from a parse tree.

4.5 Automatic Generation of an Automatic Grammar File Generator

The solution outlined in the previous section for producing custom grammar files is not entirely satisfactory. If the scope of the general grammar file is changed or the notation of the grammar file is switched from, say, Mathematica notation to FrameMaker notation then the grammar file generator (the transcription parser) must be rewritten. Therefore, the goal is to automatically generate the transcription parser from the general grammar file. See Figure 4-5.

The general grammar file describes both a set of strings as well as the corresponding images associated with those strings. The equation parser tries to reconstruct the parse tree from the image. However, one can also construct a parser for the strings that the general grammar file describes. These strings are exactly the transcription strings. Therefore, using the information in the general grammar file a transcription parser can be constructed. For example, consider the general

grammar file previously discussed in Section 2.3:

$$\begin{aligned}
\langle Etop \rangle &\rightarrow \langle Etopleft \rangle \langle Eplus \rangle &&: 0.75 \quad \text{“\%1\%2]”} \\
\langle Etop \rangle &\rightarrow \langle Eplus \rangle &&: 0.25 \quad \text{“\%1”} \\
\langle Etopleft \rangle &\rightarrow \langle Eplus \rangle \text{ ‘=’} &&: 1.0 \quad \text{“equal[\%1,”} \\
\langle Eplus \rangle &\rightarrow \langle Eplusleft \rangle \langle T \rangle &&: 0.5 \quad \text{“\%1\%2]”} \\
\langle Eplus \rangle &\rightarrow \langle T \rangle &&: 0.5 \quad \text{“\%1”} \\
\langle Eplusleft \rangle &\rightarrow \langle Eplus \rangle \text{ ‘+’} &&: 1.0 \quad \text{“plus[\%1,”} \\
\langle T \rangle &\rightarrow \text{‘a’} &&: 0.3 \quad \text{“a”} \\
\langle T \rangle &\rightarrow \text{‘b’} &&: 0.3 \quad \text{“b”} \\
\langle T \rangle &\rightarrow \text{‘c’} &&: 0.4 \quad \text{“c”}
\end{aligned}$$

This general grammar file describes a set of transcription strings from a context-free language. Some example transcription strings are “Times[a,b]”, “Equal[b,c]”, “Equal[Times[a,c],b]”, “Equal[Times[Times[a,a],a], Times[b,c]].” A grammar file producing these string can be written as follows:

$$\begin{aligned}
\langle S \rangle &\rightarrow \langle Eequal \rangle \\
\langle Eequal \rangle &\rightarrow \text{‘equal[’ } \langle Eplus \rangle \text{ ‘,’ } \langle Eplus \rangle \text{ ‘]’} \\
\langle Eequal \rangle &\rightarrow \langle Eplus \rangle \\
\langle Eplus \rangle &\rightarrow \text{‘plus[’ } \langle Eplus \rangle \text{ ‘,’ } \langle T \rangle \text{ ‘]’} \\
\langle Eequal \rangle &\rightarrow \langle T \rangle \\
\langle T \rangle &\rightarrow \text{‘a’} \\
\langle T \rangle &\rightarrow \text{‘b’} \\
\langle T \rangle &\rightarrow \text{‘c’.}
\end{aligned}$$

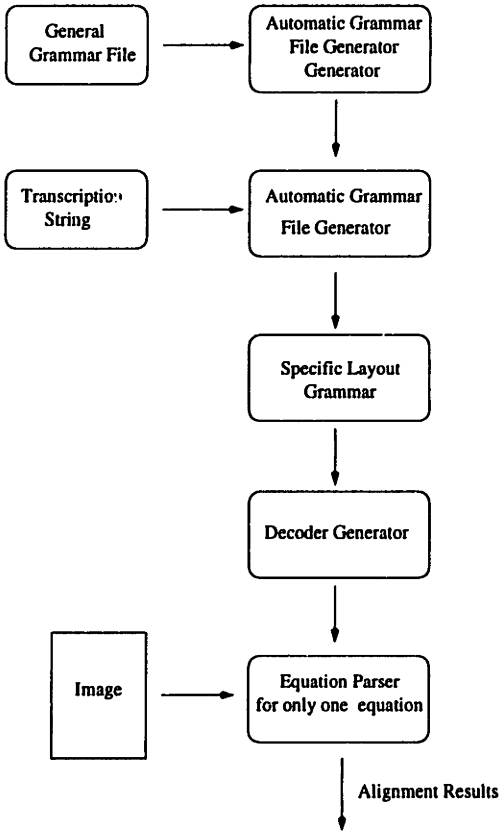


Figure 4-5: Block diagram for automatic generation of transcription parser.

Chapter 5

Learning of Parameters in Layout Model

5.1 Introduction

The overall approach to supervised learning is depicted in Figure 5-1. The set of equations to be recognized is divided into two sets – the training data and the test data. For each equation in the training data a transcription is provided. Using this transcription the decoder performs an initial alignment on the training data as described in Chapter 4. From this information various parameters in the layout model are estimated, increasing its accuracy. The decoder then uses this new layout model to perform recognition on the test data. This cycle of recognizing, estimating parameters, then recognizing is repeated until all the parameters stabilize. Unsupervised learning, in which a transcription is not provided with the training data is also possible but not discussed here.

5.2 Maximum Likelihood Estimation of Parameters

Recall that when the decoder performs recognition the goal is to find the message $\hat{M}$ that maximizes the posterior probability of the message $\hat{M}$ given the image O, namely to maximize $P(\hat{M}|O)$. When training is performed after with supervision,

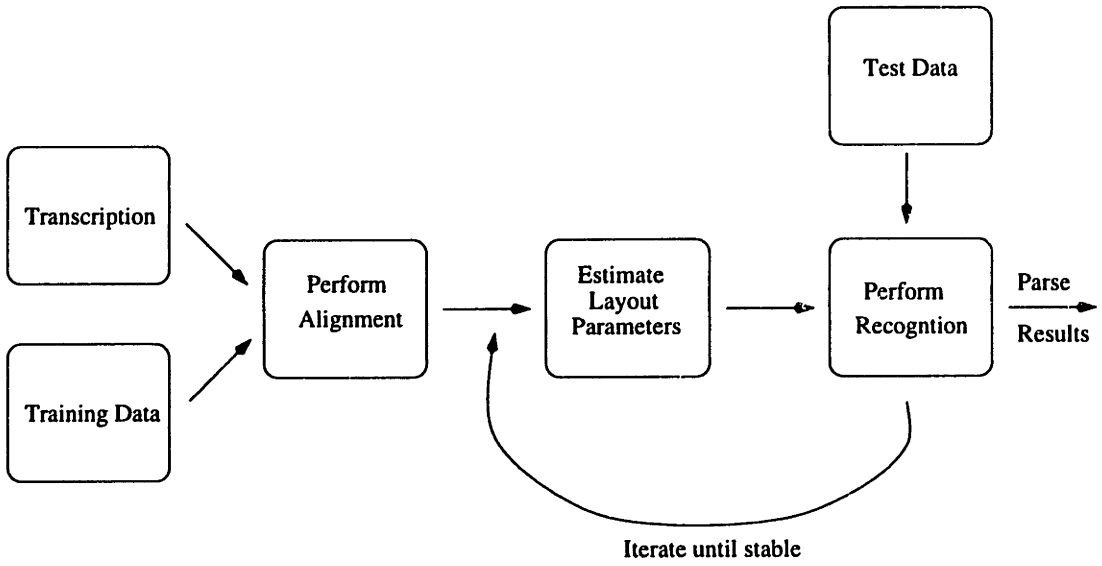


Figure 5-1: Overall approach to estimation of parameters.

the goal is to find the set of parameters in the model that maximize $P(O, \hat{M})$ for a given M and O. This set of parameters is called the maximum likelihood estimate. More generally, there will be multiple training examples, thus the maximum likelihood estimate will be

$$\underset{\text{Model Parameters}}{\arg\max} \prod_i P(\hat{M}_i, O_i), \tag{5.1}$$

where the index i ranges over training examples.

A wide number of facets in this model can be estimated. The focus shall be on learning the layout functions T_{21}, F_0 and T_{10}. However, a brief description of how to calculate maximum likelihood estimates for the grammar rule probabilities and the channel parameters is presented. For a discussion of estimation of font templates see [12].

5.3 Estimation of Grammar Rule Probabilities

Combining (3.1) and (5.1) gives

$$\underset{\text{Model Parameters}}{\arg\max} \prod_i P(\hat{M}_i, O_i) = \underset{\text{Model Parameters}}{\arg\max} \prod_i \left(\sum_I P(M_i) P(I|M_i) P(O_i|I) \right), \tag{5.2}$$

and $\sum_I P(M_i)P(I|M_i)P(O_i|I)$ is approximated by $P(M_i)P(\hat{I}_i|M_i)P(O_i|\hat{I}_i)$ where $\hat{I}_i$ maximizes $P(M_i)P(I|M_i)P(O_i|I)$. Therefore, (5.2) becomes

$$\underset{\text{Model Parameters}}{\arg\max} \prod_i P(\hat{M}_i, O_i) \approx \underset{\text{Model Parameters}}{\arg\max} \prod_i P(M_i) P(\hat{I}_i|M_i) P(O_i|\hat{I}_i). \tag{5.3}$$

Since the grammar rule probabilities only effect $P(M_i)$, it suffices to find

$$\underset{\text{Grammar Rule Probabilities}}{\arg\max} \prod_i P(M_i). \tag{5.4}$$

Combining this with equation (2.1) yields

$$\underset{p(r_1),p(r_2),\ldots,p(r_m)}{\arg\max} \prod_i \prod_j p(r_{k_{ij}}), \tag{5.5}$$

where $r_{k_{ij}}$ is the jth rule used to generate the ith message. For simplicity, assume the grammar is of the following form:

$$\begin{aligned} r_0 &: E_0 \rightarrow \ldots \\ r_1 &: E_0 \rightarrow \ldots \\ r_2 &: E_1 \rightarrow \ldots \\ r_3 &: E_1 \rightarrow \ldots \\ &\vdots \\ r_{2n} &: E_n \rightarrow \ldots \\ r_{2n+1} &: E_n \rightarrow \ldots, \end{aligned}$$

namely that every non-terminal has exactly two production rules associated with it. Although, in general, the grammar files will not be of this form the general case

follows straightforwardly. Rearranging the terms in (5.5) gives

$$\arg\max_{p(r_1),p(r_2),\ldots,p(r_m)} \prod_i p(r_i)^{a_i}, \tag{5.6}$$

where a_i is the number of times the rule r_i is used in all the of the training examples. The goal is now to find the $p(r_i)$ that maximizes (5.6) subject to the constraint that all the probabilities for a particular nonterminal sum to 1 or, equivalently,

$$p_{2i} + p_{2i+1} = 1 \quad \text{for all } 0 \leq i \leq n. \tag{5.7}$$

Solving this maximization problem using simple calculus produces

$$p_i = \begin{cases} \frac{a_i}{a_i+a_{i+1}} & i \text{ even}, \\ \frac{a_i}{a_{i-1}+a_i} & i \text{ odd}, \end{cases} \tag{5.8}$$

which says that the approximately maximum likelihood estimates for the grammar rule probabilities are proportional to their frequencies in the training data.

5.4 Estimation of Channel Parameters

Recall from Section 2.5.2 that the model of the channel only effects the $P(O_i|I_i)$ term in (5.3). Therefore, the approximate maximum likelihood estimate of the channel parameters is

$$\arg\max_{\text{channel parameters}} \prod_i P(O_i|\hat{I}_i).$$

Furthermore, Section 2.5.2 describes two basic channel parameters ϵ_0 and ϵ_1. ϵ_0 is the probability that a white pixel in the ideal image appears black in the observed image and conversely, ϵ_1 is the probability that a black pixel in the ideal image

appears white in the observed image. Letting

$$\begin{aligned}
a &= \text{number of points } (j,k) \text{ where } \hat{I}_i^{(j,k)} = 0 \text{ and } O_i^{(j,k)} = 1 \\
b &= \text{number of points } (j,k) \text{ where } \hat{I}_i^{(j,k)} = 0 \text{ and } O_i^{(j,k)} = 0 \\
c &= \text{number of points } (j,k) \text{ where } \hat{I}_i^{(j,k)} = 1 \text{ and } O_i^{(j,k)} = 0 \\
d &= \text{number of points } (j,k) \text{ where } \hat{I}_i^{(j,k)} = 1 \text{ and } O_i^{(j,k)} = 1,
\end{aligned}$$

gives, using (2.14).

$$\underset{\text{channel parameters}}{\arg\max} \prod_i P(O_i|\hat{I}_i) = \underset{\epsilon_0,\epsilon_1}{\arg\max} \left(\epsilon_0^a(1-\epsilon_0)^b\epsilon_1^c(1-\epsilon_1)^d\right).$$

And using simple calculus

$$\begin{aligned}
\epsilon_0 &= \frac{a}{a+b} \\
\epsilon_1 &= \frac{c}{c+d}.
\end{aligned}$$

Thus, as in the estimation of rule probabilities, the maximum likelihood estimate of the the channel parameters is also proportional to their frequency in the training data.

5.5 Estimating the Layout Functions

The estimation of the layout functions is done in three phases. First, the T_{21} functions are estimated while holding the T_{10} and F_0 functions constant. Next, the F_0 functions are estimated while holding the T_{21} and T_{10} functions constant. Finally, the T_{10} functions are estimated while holding the T_{21} and F_0 functions constant. This three phase processes is repeated until the parameters stabilize. Since the layout model only effects $P(I|M)$, (5.3) becomes

$$\begin{aligned}
\underset{\text{layout parameters}}{\arg\max} \prod_i P(\hat{I}_i|M_i) &= \underset{\text{layout parameters}}{\arg\max} \prod_i \sum_{T_i^\alpha : I_i(T_i^\alpha)=I_i} P(T_i^\alpha|M_i) \\
&\approx \underset{\text{layout parameters}}{\arg\max} \prod_i \max_{T_i^\alpha : I(T_i^\alpha)=I} P(T_i^\alpha|M_i) \qquad (5.9) \\
&= \underset{\text{layout parameters}}{\arg\max} \prod_i P(\hat{T}_i^\alpha|M_i),
\end{aligned}$$

where, as before, $\hat{T}_i^\alpha$ is the T_i^α that maximizes $P(T_i^\alpha|M_i)$ given that $I_i(T_i^\alpha) = I_i$. Using (2.11) the quantity being maximized is now given by

$$\underset{\text{layout parameters}}{\arg\max} \prod_i \prod_j p_{E_{ij}}(r_{k_{ij}}), \tag{5.10}$$

where E_{ij} is the error transformation observed while applying the jth rule in the message string M_i and $p_{E_{ij}}(r_{k_{ij}})$ is the probability of this error transformation. Furthermore, $p_{E_{ij}}(r)$ has a Gaussian distribution with unit variance on each of its components. The assumption that $p_{E_{ij}}(r)$ has unit variance on all its components is made to simplify the discussion. See Section 5.5.3 for a description of how the variance can be estimated. Since $p_{E_{ij}}(r)$ is Gaussian,

$$\begin{aligned} \prod_i \prod_j p_{E_{ij}}(r) &= \prod_i \prod_j \frac{1}{2\pi^{\frac{3}{2}}} e^{-\frac{1}{2}E_{ij}^T E_{ij}} \\ &= \frac{1}{(2\pi)^{\frac{3n}{2}}} e^{-\frac{1}{2}\sum_i \sum_j E_{ij}^T E_{ij}}, \end{aligned} \tag{5.11}$$

where $E_{ij}^T = [E_{ij}^{shift_x}, E_{ij}^{shift_y}, E_{ij}^{scale}]$. Furthermore, since the only term in (5.11) affected by the layout model is the error E_{ij}, it follows that

$$\underset{\text{layout parameters}}{\arg\max} \frac{1}{(2\pi)^{\frac{3n}{2}}} e^{-\frac{1}{2}\sum_i \sum_j E_{ij}^T E_{ij}} = \underset{\text{layout parameters}}{\arg\min} \sum_i \sum_j E_{ij}^T E_{ij}. \tag{5.12}$$

This means the maximum likelihood estimate of the layout parameters is the set of parameters that minimize the total mean squared error. The details of the parameterization and estimation of T_{21}, F_0, and T_{10} are now described.

5.5.1 Parameterization of the T_{21} Functions

As was stated in Section 2.4.3 the T_{21} for a rule of the form $A \rightarrow BC$ is a function of F_B and F_C – the box metrics for nodes B and C respectively. In general all three components of T_{21} could be a nonlinear function of all the components of F_B and

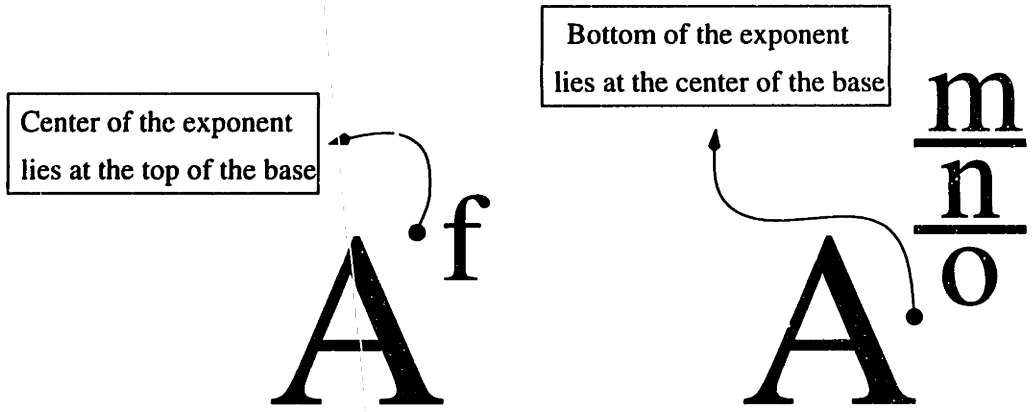


Figure 5-2: Example of a nonlinear layout rule.

F_C. However, T_{21} is restricted to be of the form

$$\frac{T_{21}^{shift_x}}{T_{21}^{scale}} = a_0 F_B^{ll_x} + a_1 F_C^{ll_x} + a_2 F_B^{upr_x} \tag{5.13}$$
$$+ a_3 F_C^{upr_x} + a_4 F_B^{end_x} + a_5 F_C^{end_x} + a_6$$
$$\frac{T_{21}^{shift_y}}{T_{21}^{scale}} = b_0 F_B^{ll_y} + b_2 F_C^{ll_y} + b_2 F_B^{upr_y} \tag{5.14}$$
$$+ b_3 F_C^{upr_y} + b_4 F_B^{end_y} + b_5 F_C^{end_y} + b_6$$
$$\log T_{21}^{scale} = c_0, \tag{5.15}$$

where $a_0, \ldots, a_6, b_0, \ldots, b_6, c_0$ are the parameters to be estimated. This form captures most of the types of equation layout one encounters. However, one does occasionally encounter nonlinear layout rules. For example, as shown in Figure 5-2, an exponentiation rule might specify that the center of the exponent (f) must lie at the top of the base (A) unless the exponent will fall below the baseline, in which case, the bottom of the exponent ($\frac{m}{\frac{n}{o}}$) must lie half way up the base. However, for simplicity these types of rules are ignored.

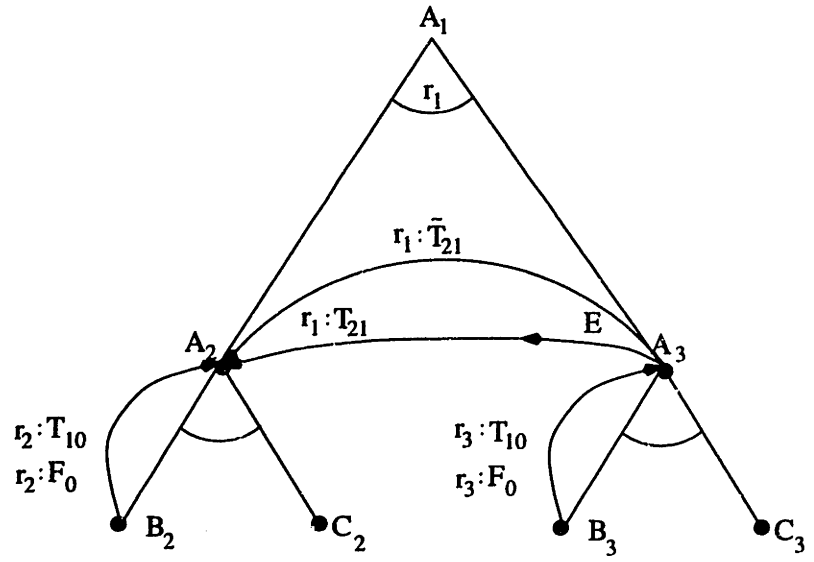


Figure 5-3: Relation between various layout functions.

5.5.2 Estimation of the T_{21} Functions

Recall (2.10) from Section 2.4.5 (2.10)

$$\tilde{T}_{21} = T_{21}(F_B, F_C) \circ E,$$

where T_{21} and $\tilde{T}_{21}$ are the predicted and observed T_{21} transformations, respectively. Solving for E gives

$$E = (T_{21})^{-1}\tilde{T}_{21}. \tag{5.16}$$

Since the mean squared error of E is being minimized, (5.16) says that an observation of $\tilde{T}_{21}$ constrains T_{21}. For example, consider Figure 5-3. When predicting T_{21}, $r_1 : \tilde{T}_{21}$ is observed, placing a restriction on the corresponding $r_1 : T_{21}$. Therefore, each $r_1 : \tilde{T}_{21}$ observed corresponds to one equation for $r_1 : T_{21}$. Hence, one can collect together all the equations involving $r_1 : T_{21}$ and solve them. In this way each $r_i : T_{21}$ can be solved for individually. See Figure 5-4. Collecting together the instances of (2.10) for each rule r (as in Figure 5-4), there is a set of equations that

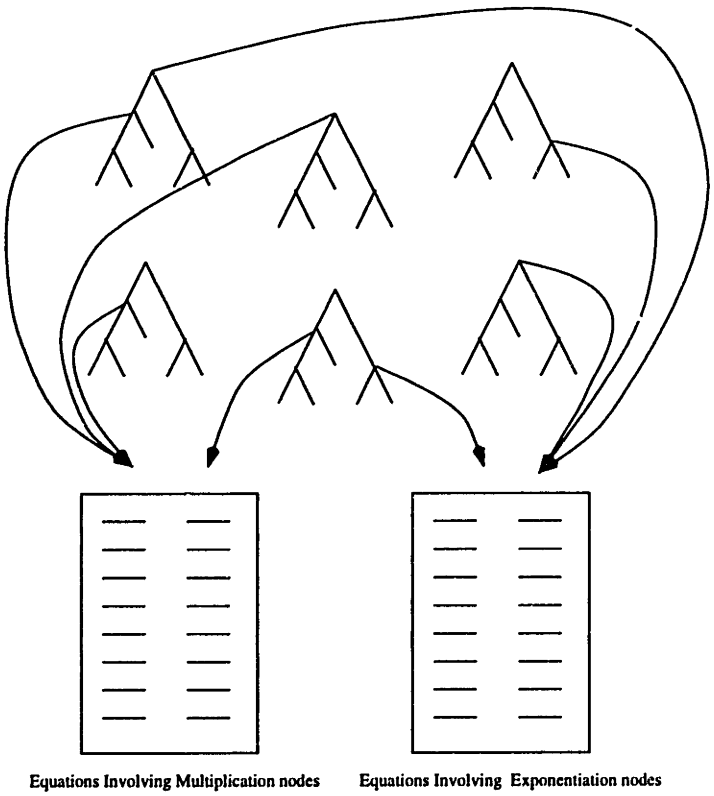


Figure 5-4: Collecting together instances of multiplication and exponentiation rules.

look like:

$$E_1 = (T_{21}^1)^{-1} \circ \tilde{T}_{21}^1 \tag{5.17}$$

$$E_2 = (T_{21}^2)^{-1} \circ \tilde{T}_{21}^2 \tag{5.18}$$

$$\vdots$$

$$E_n = (T_{21}^n)^{-1} \circ \tilde{T}_{21}^n, \tag{5.19}$$

where T_{21}^i and $\tilde{T}_{21}^i$ are the predicted and observed T_{21} transformations for the ith use of the rule r in the training data. Similarly, E_i is the observed error transformation for the ith use of the rule r in the training data. (Note that this is a change for the previous notation of E_{ij}). With this new notation, for each rule r

$$\sum_j E_j^T E_j$$

is being minimized. Letting

$$\epsilon^T = [E_1^{shift_x}, E_2^{shift_x}, \ldots, E_n^{shift_x}, E_1^{shift_y}, \ldots, E_n^{shift_y}, E_1^{lscale}, \ldots, E_n^{lscale}],$$

it suffices to minimize $\epsilon^T\epsilon$. Finally, the equations for E in component form are

$$E^{shift_x} = \frac{\tilde{T}_{21}^{shift_x} - T_{21}^{shift_x}}{T_{21}^{scale}} \tag{5.20}$$

$$E^{shift_y} = \frac{\tilde{T}_{21}^{shift_y} - T_{21}^{shift_y}}{T_{21}^{scale}} \tag{5.21}$$

$$E^{lscale} = \log(\tilde{T}_{21}^{scale}) - \log(T_{21}^{scale}) \tag{5.22}$$

Combining (5.13)–(5.15), (5.17)–(5.19), and (5.20)–(5.22) gives

$$A\vec{x} = \vec{b} - \epsilon \tag{5.23}$$

where

$$A = \begin{bmatrix}
{}_1F_B^{ll_x} & {}_1F_C^{ll_x} & {}_1F_B^{upr_x} & {}_1F_C^{upr_x} & {}_1F_B^{end_x} & {}_1F_C^{end_x} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\ 0 \\
{}_2F_B^{ll_x} & {}_2F_C^{ll_x} & {}_2F_B^{upr_x} & {}_2F_C^{upr_x} & {}_2F_B^{end_x} & {}_2F_C^{end_x} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\ 0 \\
 & & & & \vdots & & & & & & & & & \\
{}_nF_B^{ll_x} & {}_nF_C^{ll_x} & {}_nF_B^{upr_x} & {}_nF_C^{upr_x} & {}_nF_B^{end_x} & {}_nF_C^{end_x} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\ 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & {}_1F_B^{ll_y} & {}_1F_C^{ll_y} & {}_1F_B^{upr_y} & {}_1F_C^{upr_y} & {}_1F_B^{end_y} & {}_1F_C^{end_y} & 1\ 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & {}_2F_B^{ll_y} & {}_2F_C^{ll_y} & {}_2F_B^{upr_y} & {}_2F_C^{upr_y} & {}_2F_B^{end_y} & {}_2F_C^{end_y} & 1\ 0 \\
 & & & & \vdots & & & & & & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & {}_nF_B^{ll_y} & {}_nF_C^{ll_y} & {}_nF_B^{upr_y} & {}_nF_C^{upr_y} & {}_nF_B^{cnd_y} & {}_nF_C^{end_y} & 1\ 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\ 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\ 1 \\
 & & & & \vdots & & & & & & & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\ 1
\end{bmatrix},$$

$$\vec{x} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ c_0 \end{bmatrix},$$

and

$$\vec{b} = \begin{bmatrix} {}_1\tilde{T}_{21}^{shift_x}/{}_1\tilde{T}_{21}^{scale} \\ {}_2\tilde{T}_{21}^{shift_x}/{}_2\tilde{T}_{21}^{scale} \\ \vdots \\ {}_n\tilde{T}_{21}^{shift_x}/{}_n\tilde{T}_{21}^{scale} \\ {}_1\tilde{T}_{21}^{shift_y}/{}_1\tilde{T}_{21}^{scale} \\ {}_2\tilde{T}_{21}^{shift_y}/{}_2\tilde{T}_{21}^{scale} \\ \vdots \\ {}_n\tilde{T}_{21}^{shift_y}/{}_n\tilde{T}_{21}^{scale} \\ \log({}_1\tilde{T}_{21}^{scale}) \\ \log({}_2\tilde{T}_{21}^{scale}) \\ \vdots \\ \log({}_n\tilde{T}_{21}^{scale}) \end{bmatrix}.$$

There is one of these equations for each type of rule. This corresponds to learning one type of T_{21} function for each type of layout rule in the grammar. These equations are solved iteratively. First $\hat{x}$ is estimated then $\hat{b}$ is estimated based on the new value of $\tilde{T}_{21}^{scale}$. This process continues until all the parameters stabilize. However, because this second level iteration is being carried out inside an overall iteration (estimating T_{21}, F_0, then T_{10}) $\hat{x}$ is only estimated once for each overall iteration. See Appendix B for the numerical technique used to solve this equation.

5.5.3 Estimating the Variance of the Error

Previously, it was assumed that E has a Gaussian distribution with unit variance in each component. Here, E is allowed to have an arbitrary 3 by 3 covariance matrix Λ_r, where Λ_r is specific to the rule r. Furthermore, Λ_r shall be estimated based on the training data. Hence, the probability density for E is of the form

$$p_E(r) = \frac{1}{2\pi^{\frac{3}{2}}}|\Lambda_r|^{-\frac{1}{2}}e^{-\frac{1}{2}E^T\Lambda_r^{-1}E}, \tag{5.24}$$

where Λ_r is the covariance matrix for the rule r. Although it shall not be proved here, the maximum likelihood estimated of $\vec{x}$ in this more general case is still given by the linear least squares solution to (5.23). Furthermore, the maximum likelihood estimate of Λ_r is given by

$$\Lambda_r = VV^T, \tag{5.25}$$

where

$$V = \begin{bmatrix} E_1^{shift_x} & E_2^{shift_x} & \cdots & E_n^{shift_x} \\ E_1^{shift_y} & E_2^{shift_y} & \cdots & E_n^{shift_y} \\ \log(E_1^{scale}) & \log(E_2^{scale}) & \cdots & \log(E_n^{scale}) \end{bmatrix}.$$

See [18] for a more complete treatment of maximum likelihood estimation.

5.5.4 Parameterization of the F_0 Functions

Recall from Section 2.4.3 that the F_0 function for a rule of the form $A \rightarrow BC$ or $A \rightarrow B$ is the function that computes the box metrics for the internal node A. When the rule is of the form $A \rightarrow BC$, F_0 takes as arguments the box metrics for B and C *in the coordinate system of* A, which are given by the expressions $\bar{F}_B = T_{10} \circ F_B$ and $\bar{F}_C = T_{10} \circ (T_{21} \circ F_C)$, respectively. Here F_B and F_C are the box metrics for the nodes B and C in their own coordinate system. When the rule is of the form $A \rightarrow B$, F_0 only depends on $\bar{F}_B$. Furthermore, the first two vectors of the box metrics for any node are simply the lower left and upper right coordinates for that node. This means that these two vectors are always calculated in the same manner. Therefore, when parameterizing the F_0 function for each rule in the grammar one only need supply information about how to calculate the end point. Hence, for rules of the form $A \rightarrow BC$, F_0 is parameterized as

$$\begin{aligned} F_0^{end_x} = {} & d_0 \bar{F}_B^{ll_x} + d_1 \bar{F}_C^{ll_x} + d_2 \bar{F}_B^{upr_x} \\ & + d_3 \bar{F}_C^{upr_x} + d_4 \end{aligned} \tag{5.26}$$

$$\begin{aligned} F_0^{end_y} = {} & e_0 \bar{F}_B^{ll_y} + e_1 \bar{F}_C^{ll_y} + e_2 \bar{F}_B^{upr_y} \\ & + e_3 \bar{F}_C^{upr_y} + e_4. \end{aligned} \tag{5.27}$$

For rules of the form $A \rightarrow B$, F_0 should, in principle, be a linear function of all the components of $\bar{F}_B$. However, in practice, the F_0 function for rules of the form $A \rightarrow B$ is usually

$$F_0^{end_x} = \bar{F}_B^{end_x} \tag{5.28}$$

$$F_0^{end_y} = \bar{F}_B^{end_y}. \tag{5.29}$$

The one exception to this rule occurs when B is a terminal symbol and the font metrics (namely end_x and end_y) of B are not known. Therefore, the F_0 functions for rules of the form $A \rightarrow B$ shall only be parameterized when B is a terminal symbol. In this case, F_0 is parameterized as

$$F_0^{end_x} = d_0 \tag{5.30}$$

$$F_0^{end_y} = e_0 \tag{5.31}$$

This corresponds to learning the end point of the font metrics for B.

5.5.5 Estimation of the F_0 Function

The estimation of the F_0 and T_{10} functions is significantly more difficult because these functions, unlike T_{21}, can not be estimated on a rule by rule basis. When solving for F_0 and T_{10}, it is assumed that the T_{21} function is constant. Consequently, the only way to reduce the mean square error is to change the arguments to the T_{21} function. In the case of Figure 5-3, this means changing F_{A_2} and F_{A_3}. These two values are determined by the layout functions $r_2 : F_0$, $r_2 : T_{10}$, $r_3 : F_0$, and $r_3 : F_{10}$. Therefore, the constraint that $E^T E$ must be minimized introduces an equation involving the layout functions for both rules r_2 and r_3. This means that there is one large linear least squares problem where all the parameters of every F_0 function (and later every T_{10} function) are solved for simultaneously.

As in Section 5.5.4, it is assumed that the F_0 function for rules of the form $A \rightarrow B$ is given by (5.28) and (5.29), unless B is a terminal symbol. When this is true, the types of nodes that must be dealt with when training the F_0 functions are shown in Figure 5-5. In case I both the children of A_1 are binary nodes. In cases II + III one of A_1's children is a terminal node. In case IV both the children of A_1 are terminal nodes. Since the F_0 function for terminal nodes is parameterized

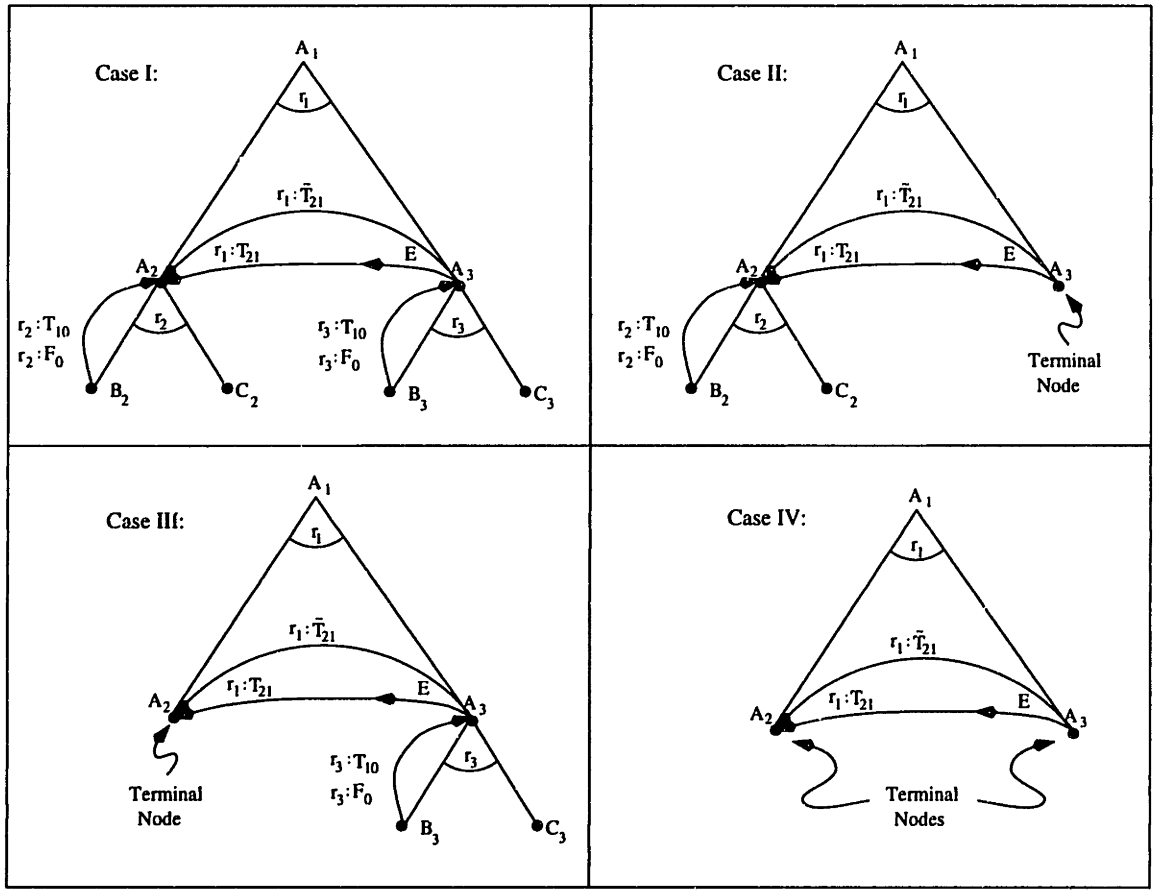


Figure 5-5: Four cases to consider when estimating the F_0 function.

differently, each of these four cases must handled separately. The four cases shall be described in reverse order.

In case IV both A_2 and A_3 are terminal symbols. Therefore, the parameterization of both $r_2 : F_0^{end_x}$ and $r_3 : F_0^{end_x}$ are described by (5.30). Hence, there are now two d_0 coefficients which shall be denoted $d_0^{r_2}$ and $d_0^{r_3}$. Combining (5.13) and (5.30) gives

$$\frac{T_{21}^{shift_x}}{T_{21}^{scale}} = a_0 F_{A_2}^{ll_x} + a_1 F_{A_3}^{ll_x} + a_2 F_{A_2}^{upr_x} + a_3 F_{A_3}^{upr_x} + a_6 + a_4 d_0^{r_2} + a_5 d_0^{r_3}. \qquad (5.32)$$

Note that only the terms for $F_{A_2}^{end_x}$ and $F_{A_3}^{end_x}$ have changed from (5.13). Collecting together all the constant terms in (5.32) and combining them with (5.17) gives

(remembering that the T_{21} coefficients $a_0, \ldots, a_6$ are now constant)

$$K - E^{shift_x} = d_0^{r_2} a_4 + d_0^{r_3} a_5, \tag{5.33}$$

where

$$\begin{aligned} K = \frac{\tilde{T}_{21}^{shift_x}}{T_{21}^{scale}} &- a_0 F_{A_2}^{ll_x} - a_1 F_{A_3}^{ll_x} - a_2 F_{A_2}^{upr_x} \\ &- a_3 F_{A_3}^{upr_x} - a_6. \end{aligned} \tag{5.34}$$

Since cases II and III are symmetric only case II is presented. Case III is similar. In case II, A_3 is still a terminal node thus $r_3 : F_0^{end_x}$ is again described by (5.30). However, A_2 is a binary node, hence, $r_2 : F_0^{end_x}$ is given by (5.26). Therefore the equivalent of (5.32) for case II is obtained by combining (5.13), (5.26), and (5.30), yielding

$$\begin{aligned} \frac{T_{21}^{shift_x}}{T_{21}^{scale}} = {} & a_0 F_{A_2}^{ll_x} + a_1 F_{A_3}^{ll_x} + a_2 F_{A_2}^{upr_x} + a_3 F_{A_3}^{upr_x} + a_6 \\ & + a_4 \begin{bmatrix} d_0^{r_2} \bar{F}_{B_2}^{ll_x} + d_1^{r_2} \bar{F}_{C_2}^{ll_x} + d_2^{r_2} \bar{F}_{B_2}^{upr_x} \\ + d_3^{r_2} \bar{F}_{C_2}^{upr_x} + d_4^{r_2} \end{bmatrix} \\ & + a_5 d_0^{r_3}. \end{aligned} \tag{5.35}$$

As in case IV, this is combined with (5.17), this time giving

$$\begin{aligned} K - E^{shift_x} = {} & d_0^{r_2} (\bar{F}_{B_2}^{ll_x} a_4) + d_1^{r_2} (\bar{F}_{C_2}^{ll_x} a_4) + d_2^{r_2} (\bar{F}_{B_2}^{upr_x} a_4) \\ & + d_3^{r_2} (\bar{F}_{C_2}^{upr_x} a_4) + d_4^{r_2} a_4 + d_0^{r_3} a_5, \end{aligned} \tag{5.36}$$

where

$$\begin{aligned} K = \frac{\tilde{T}_{21}^{shift_x}}{T_{21}^{scale}} &- a_0 F_{A_2}^{ll_x} - a_1 F_{A_3}^{ll_x} - a_2 F_{A_2}^{upr_x} \\ &- a_3 F_{A_3}^{upr_x} - a_6. \end{aligned} \tag{5.37}$$

Case I is similar to cases II and III except that both A_2 and A_3 are nonterminals. Hence, both functions $r_2 : F_0^{end_x}$ and $r_3 : F_0^{end_x}$ are parameterized by (5.26). Thus,

the equivalent of (5.32) and (5.35) for case I is

$$\begin{aligned}\frac{T_{21}^{shift_x}}{T_{21}^{scale}} &= a_0 F_{A_2}^{ll_x} + a_1 F_{A_3}^{ll_x} + a_2 F_{A_2}^{upr_x} \\ &\quad + a_3 F_{A_3}^{upr_x} + a_6 \\ &\quad + a_4 \begin{bmatrix} d_0^{r2} \bar{F}_{B_2}^{ll_x} + d_1^{r2} \bar{F}_{C_2}^{ll_x} + d_2^{r2} \bar{F}_{B_2}^{upr_x} \\ + d_3^{r2} \bar{F}_{C_2}^{upr_x} + d_4^{r2} + U_x^{r2}(\bar{F}_{B_2}^{end_x}, \bar{F}_{C_2}^{end_x}) \end{bmatrix} \\ &\quad + a_5 \begin{bmatrix} d_0^{r3} \bar{F}_{B_3}^{ll_x} + d_1^{r3} \bar{F}_{C_3}^{ll_x} + d_2^{r3} \bar{F}_{B_3}^{upr_x} \\ + d_3^{r3} \bar{F}_{C_3}^{upr_x} + d_4^{r3} + U_x^{r3}(\bar{F}_{B_3}^{end_x}, \bar{F}_{C_3}^{end_x}) \end{bmatrix},\end{aligned} \tag{5.38}$$

and the corresponding equations to (5.33) and (5.36) are

$$\begin{aligned}K - E^{shift_x} &= d_0^{r2}(\bar{F}_{B_2}^{ll_x} a_4) + d_1^{r2}(\bar{F}_{C_2}^{ll_x} a_4) + d_2^{r2}(\bar{F}_{B_2}^{upr_x} a_4) \\ &\quad + d_3^{r2}(\bar{F}_{C_2}^{upr_x} a_4) + d_4^{r2} a_4 \\ &\quad + d_0^{r3}(\bar{F}_{B_3}^{ll_x} a_5) + d_1^{r3}(\bar{F}_{C_3}^{ll_x} a_5) + d_2^{r3}(\bar{F}_{B_3}^{upr_x} a_5) \\ &\quad + d_3^{r3}(\bar{F}_{C_3}^{upr_x} a_5) + d_4^{r3} a_5,\end{aligned} \tag{5.39}$$

where

$$\begin{aligned}K &= \frac{\tilde{T}_{21}^{shift_x}}{T_{21}^{scale}} - a_0 F_{A_2}^{ll_x} - a_1 F_{A_3}^{ll_x} - a_2 F_{A_2}^{upr_x} \\ &\quad - a_3 F_{A_3}^{upr_x} - U_x^{r2}(\bar{F}_{B_2}^{end_x}, \bar{F}_{C_2}^{end_x}) - U_x^{r3}(\bar{F}_{B_3}^{end_x}, \bar{F}_{C_3}^{end_x}) - a_6.\end{aligned} \tag{5.40}$$

In all four cases the equations involving $e_0^{r2}, \ldots, e_4^{r2}, e_0^{r3}, \ldots, e_4^{r3}$ correspond exactly to those for $d_0^{r2}, \ldots, d_4^{r2}, d_0^{r3}, \ldots, d_4^{r3}$ except all x-component terms are replaced by the equivalent y-component terms.

Collecting together all the equations from cases I, II, III, and IV gives one large system of linear equations. This linear system might look like

$$\begin{bmatrix} * & * & * & * & * & \cdots & * & * & * & * & * & \cdots & * & \cdots & \cdots \\ \cdots & & & & \cdots & & * & * & * & * & * & \cdots & * & & \cdots \\ \cdots & & * & & & \cdots & & & & & & \cdots & & * & * & * & * & * \\ & & & & & & \vdots & & & & & & & & \\ \cdots & & * & & & \cdots & & \cdots & * & & & \cdots & & & \end{bmatrix} \begin{bmatrix} d_0^{r1} \\ \vdots \\ d_4^{r1} \\ d_0^{r2} \\ d_0^{r3} \\ \vdots \\ d_0^{rn} \\ \vdots \\ d_4^{rn} \end{bmatrix} = \vec{K} - \epsilon, \tag{5.41}$$

where $*$ represents non-zero terms and $\vec{K}$ is the vector of constant terms. The F_0 functions are parameterized with either one coefficient d_0 or with five coefficients $d_0, \ldots, d_4$. Therefore each row of the matrix in (5.41) either has two nonzero terms (case IV), six nonzero terms (cases II and III) or has ten nonzero terms (case I). There is a similar system of linear equations for the $e_0, \ldots, e_4$ coefficients.

5.5.6 Parameterization of the T_{10} Functions

Recall from Section 2.4.3 that the T_{10} function for a rule of the form $A \rightarrow BC$ or $A \rightarrow B$ is a coordinate transformation from the coordinate system of B to the coordinate system of A. When the rule is of the form $A \rightarrow B$ this function depends only on the box metrics of B. However, when the rule is of the form $A \rightarrow BC$, this function depends on the box metrics of B and C as well as T_{21}, the transformation between them. However, in the parameterization of T_{10} for rules of the form $A \rightarrow BC$ the dependence on T_{21} and F_C is made implicit by using only $\hat{F}_C = T_{21} \circ F_C$ in the expression for T_{10}. The parameterization of T_{10} is given by

$$\begin{aligned} T_{10}^{shift_x} &= f_0 F_B^{ll_x} + f_1 \hat{F}_C^{ll_x} + f_2 F_B^{upr_x} \\ &\quad + f_3 \hat{F}_C^{upr_x} + f_4 F_B^{end_x} + f_5 \hat{F}_C^{end_x} + f_6 \end{aligned} \tag{5.42}$$

$$\begin{aligned} T_{10}^{shift_y} &= h_0 F_B^{ll_y} + h_2 \hat{F}_C^{ll_y} + h_2 F_B^{upr_y} \\ &\quad + h_3 \hat{F}_C^{upr_y} + h_4 F_B^{end_y} + h_5 \hat{F}_C^{end_y} + h_6 \end{aligned} \tag{5.43}$$

$$\log T_{10}^{scale} = j_0. \tag{5.44}$$

The parameterization of T_{10} for rules of the form $A \rightarrow B$ should, in principle, depended on F_B. However, as in the estimation of F_0, T_{10} shall take on a trivial

form except when B is a terminal node. The trivial form for T_{10} is the identity transformation. When B is a terminal node, T_{10} is parameterized as

$$T_{10}^{shift_x} = f_0 \tag{5.45}$$

$$T_{10}^{shift_y} = h_0 \tag{5.46}$$

$$\log T_{10}^{scale} = j_0. \tag{5.47}$$

Hence, the T_{10} function for rules of the form $A \rightarrow B$, where B is a terminal node determines the origin and font size of the character that B represents.

5.5.7 Estimating the T_{10} functions

The estimation of T_{10} also falls into the four cases described in the previous section since the T_{10} function associated with terminal nodes differs from the T_{10} function associated with binary nodes. Furthermore, only case II is presented since it handles both a binary node and a terminal node. As with estimating the F_0 functions, the other cases follow straightforwardly.

The approach to estimating T_{10} is to again attempt to minimize the mean squared error of (5.35) except now, the following expressions are substituted for the terms in (5.35):

$$F_{A_2}^{ll_x} = T_{10}^{scale} \min(F_{B_2}^{ll_x}, \hat{F}_{C_2}^{ll_x}) + T_{10}^{shift_x} \tag{5.48}$$

$$F_{A_2}^{upr_x} = T_{10}^{scale} \max(F_{B_2}^{upr_x}, \hat{F}_{C_2}^{upr_x}) + T_{10}^{shift_x} \tag{5.49}$$

$$\bar{F}_{B_2} = T_{10} \circ F_{B_2} \tag{5.50}$$

$$\bar{F}_{C_2} = T_{10} \circ \hat{F}_{C_2}. \tag{5.51}$$

There is a similar set of equations for $F_{A_3}^{ll_x}, F_{A_3}^{upr_x}, \bar{F}_{B_3}, \bar{F}_{C_3}$. Combining (5.35)–(5.36), (5.42)–(5.44), (5.45)–(5.47), and (5.48)–(5.51) yields

$$\begin{aligned} K_1 - E = {} & f_0^{r_2}(F_{B_2}^{ll_x} K_2) + f_1^{r_2}(F_{C_2}^{ll_x} K_2) f_2^{r_2}(F_{B_2}^{upr_x} K_2) + f_3^{r_2}(F_{C_2}^{upr_x} K_2) \\ & + f_4^{r_2}(F_{B_2}^{end_x} K_2) + f_5^{r_2}(F_{C_2}^{end_x} K_2) + f_6^{r_2}(K_2) \\ & + f_0^{r_3}(K_3) + \\ & + j_0^{r_2}(K_4) + j_0^{r_3}(K_5) \end{aligned}$$

where

$$
\begin{aligned}
K_1 &= \frac{\tilde{T}_{21}^{shift_x}}{T_{21}^{scale}} - a_4 d_4^{r^2} - a_5 d_0^{r^3} - a_6 \\
K_2 &= a_0 + a_2 + a_4(d_0^{r^2} + d_1^{r^2} + d_2^{r^2} + d_3^{r^2}) \\
K_3 &= a_1 + a_3 \\
K_4 &= a_0 \min(F_{B_2}^{ll_x}, \hat{F}_{C_2}^{ll_x}) + a_2 \max(F_{B_2}^{upr_x}, \hat{F}_{C_2}^{upr_x}) \\
&\quad + a_4(d_0^{r^2} F_{B_2}^{ll_x} + d_1^{r^2} \hat{F}_{C_2}^{ll_x} + d_2^{r^2} F_{B_2}^{upr_x} + d_3^{r^2} \hat{F}_{C_2}^{upr_x}) \\
K_5 &= a_1 \min(F_{B_3}^{ll_x}, \hat{F}_{C_3}^{ll_x}) + a_3 \max(F_{B_3}^{upr_x}, \hat{F}_{C_3}^{upr_x}).
\end{aligned}
$$

The equations for the y-components $h_0, \ldots, h_6$ are similar. Collecting together all the equations of the above form, there is again a system of linear equations for which the linear least squares solution is found. The main difference between this case and the F_0 case is that the x-coefficients $f_0, \ldots, f_6$ and the y-coefficients $h_0, \ldots, h_6$ can no longer be solved for independently because they both involve the scale term j_0.

Chapter 6

Design of the System

6.1 Introduction

The recognition system is divided into a series of modules that communicate with each other. As can be seen in Figure 6-1 each module does part of the recognition or training task. Until now, the concentration has been on describing the recognition and training of the equations. Although this is the major focus of this work, the other aspects involved in building a recognition system are now described.

6.2 Interface to Handwritten Mathematics

The system uses three front ends: one for handwritten mathematics, one for scanned typeset mathematics, and a third for typeset mathematics in a POSTSCRIPT file. The interface for handwritten mathematics, implemented in Motif under XWindows, operates as follows. (See Figure 6-2.) The user may draw mathematical expressions using the mouse or a pen input device. The interface supports saving and loading sets of handwritten expressions as well as simple editing by adding and removing individual strokes. When editing is complete, the strokes that the user has entered are passed to the stroke capture module. The interface also has a rudimentary text display window to print the results of the character recognition and of the expression recognition. This text window is also used to display the results of passing the recognized expression to Mathematica.

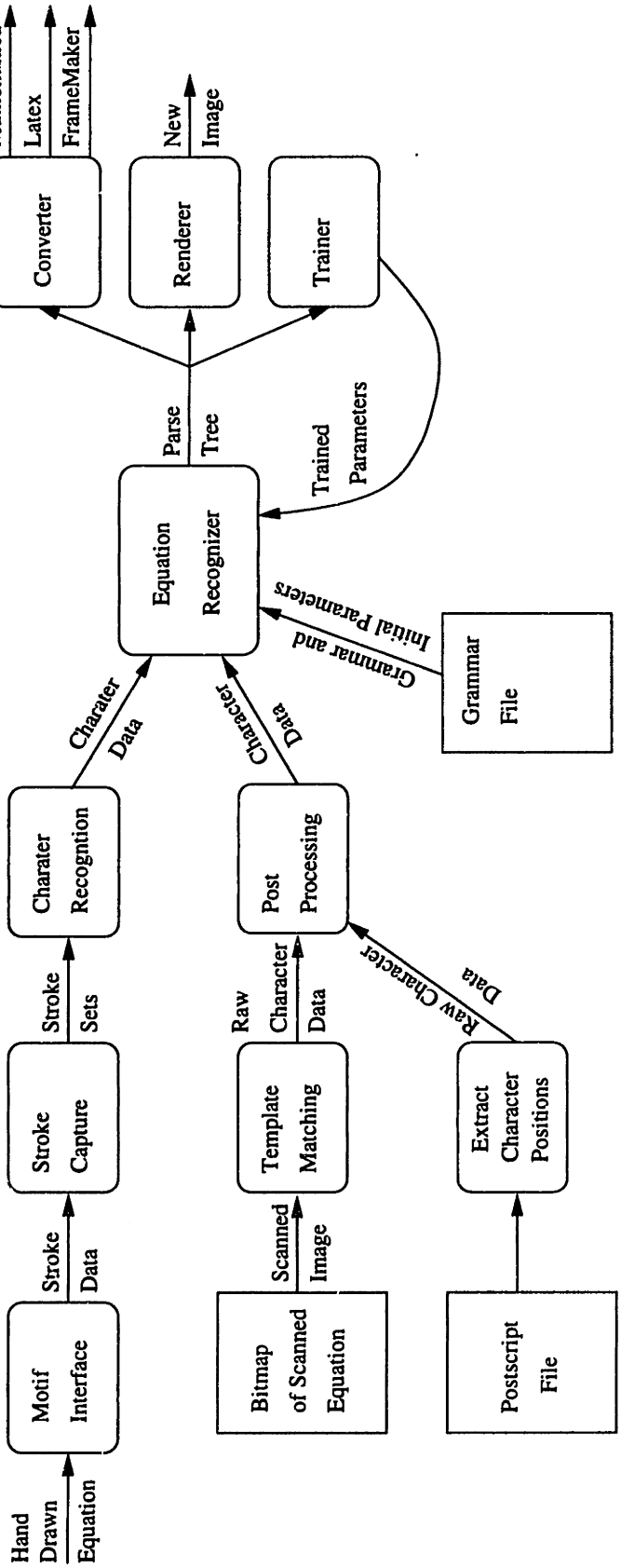


Figure 6-1: Overall block diagram of the equation recognition system.

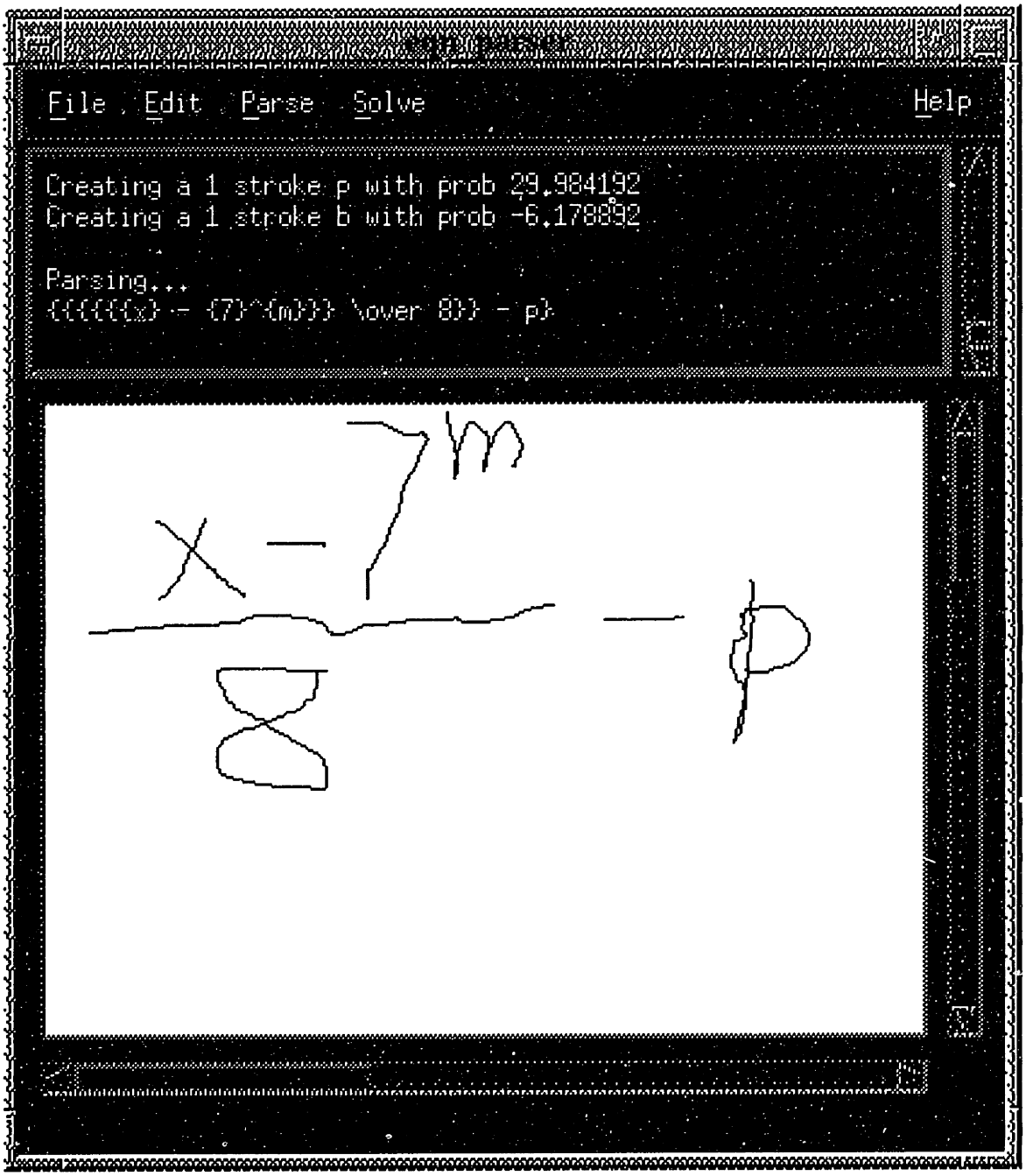


Figure 6-2: The interface to input handwritten mathematics.

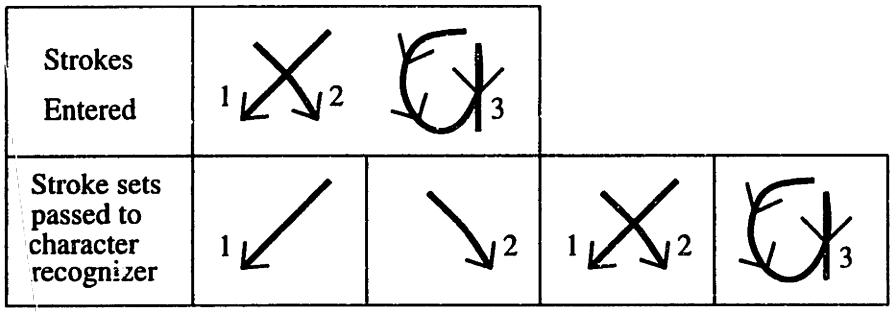


Figure 6-3: Example of stroke capture on two characters.

6.3 Stroke Capture

The stroke capture module takes the individual stroke data and determines which sets of strokes are likely to represent characters. Two strokes are considered to be likely to represent a single character if the strokes overlap and are drawn consecutively. See Figure 6-3. The stroke capture module also normalizes the size of any characters passed to the recognizer.

6.4 Handwritten Character Recognition

The handwritten character recognition module was originally implemented using Nestor, a commercially available character recognizer. Nestor is lacking in two ways. First, it performs recognition using only bitmaps, rather then using the position of the pen recorded by the front end. This position information can certainly be used to generate a bitmap, but this discards information that could help with recognition. Second, Nestor returns the top five candidate characters sorted by the score $P(c|O)$ where O is the observed image and c the predicted character. However, the quantity needed for equation reconstruction is (as in Chapter 3)

$$P(O|c) = \frac{P(c|O)P(O)}{P(c)}.$$

The quantity $P(c)$, the prior probability of get the character c, can be estimated fairly easily. However, there is no obvious way to estimate $P(O)$, the prior probability of drawing image O.

Thus, Nestor was replaced by a different character recognition system based on hidden Markov models. The system uses the raw character stroke data and produces for a character c a score representing $P(S|c)$, where S is the set of strokes representing the character. The system is based on a one stroke gesture recognition system developed by Tichomir Tenev [19] and was extended to handle multiple stroke characters as well as larger character sets The handwritten character recognition module passes information about the likelihood, position, and size of each character on the page to the equation recognition module.

6.5 Character Recognition for Typeset Mathematics

Character recognition for typeset equations is performed using an algorithm developed by William Rucklidge and Dan Huttenlocher. This algorithm is now described. There is one character template, a bitmap, for each character in each typeface and font size. Each template is placed at every possible position on the page. For every position, it is determined whether the template matches the image. A character template matches an image if a fraction M_{perc} of the template "on" bits are within M_{dist} of an "on" bit in the image and a fraction I_{perc} of the image on bits are within I_{dist} of an "on" bit in the model.

Consider, for example, the character in Figure 6-4. Here a model bitmap of the character "x" and an observed bitmap of the character "x" is depicted. Figure 6-4(c) shows that three image bits are one pixel away from the character model. Furthermore, Figure 6-4(d) shows that four of the model pixels are one pixel away from the image. This means that if both I_{dist} and M_{dist} equal 0, then

$$M_{perc} \leq \frac{100(57-4)}{57} \approx 93.0 \tag{6.1}$$

$$I_{perc} \leq \frac{100(56-3)}{56} \approx 94.6 \tag{6.2}$$

for the image to match the character model.

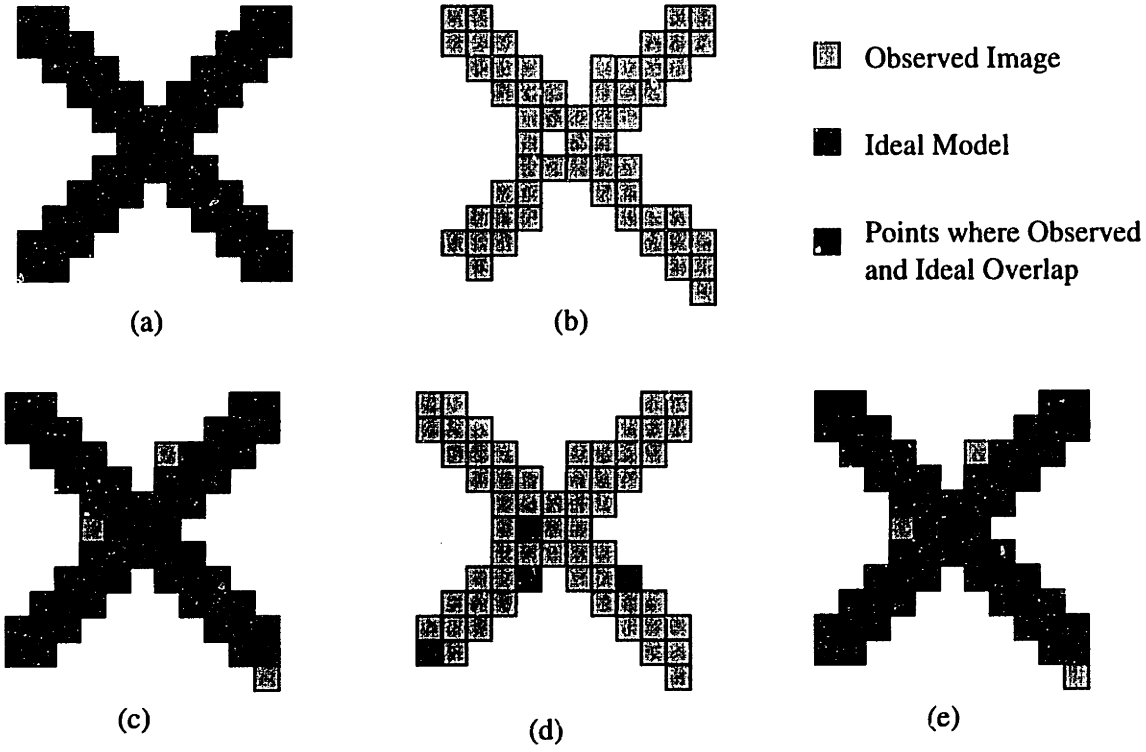


Figure 6-4: (a) a character model for "x", (b) an example image of an "x", (c) the model overlaying the image, (d) the image overlaying the model, (e) the image and model intersected

6.6 Post Processing of Template Matching Data

The data returned from the character recognizer doesn't properly handle characters that can be arbitrarily scaled. For example, the bar in a division operation has to be as long as the its longest argument. The template matcher would have to contain a different character for every possible length division bar to handle the division operation without a post processor. To avoid this problem the template matcher just returns a series of fixed length division bars. The post processing module then glues these fixed length bars together into a single division bar. A similar process is carried out for parentheses.

6.7 Extracting Character Positions from a POSTSCRIPT File

The third front end to the equation recognizer is a module that processes POSTSCRIPT files. The module takes a POSTSCRIPT file representing an equation and extracts the typeface, size, and position of every character in the POSTSCRIPT file. This module simulates perfect character recognition data. This allows testing the accuracy of the equation recognition module without having to worry about errors introduced from character recognition.

6.8 Converter

The converter module takes a representation of the final parse tree for the mathematics and converters it into a variety of formats. It can be converted into a format readable by LaTeX, Framemaker, or Mathematica. Each format corresponds to a different message grammar. Hence, to transform into any of these formats simply requires a bottom-up pass of the parse tree as is described in Section 2.3. For example, see Figure 6-5.

6.9 Renderer

The renderer takes as input a parsed equation and produces an image of that equation. The render produces this image, as described in Chapter 2, by using either the layout model that was learned during recognition, or the model that was supplied in the general grammar file. The rendered equation gives a graphical representation of the accuracy of the layout model. See Section 7.7.1 for examples of rendered equations.

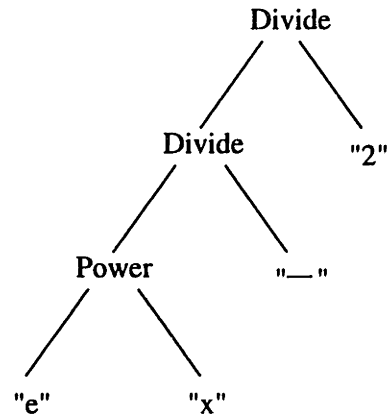
Expression	Parse Tree
$\frac{e^x}{2}$	
Mathematica format	Divide[Power[e, x], 2]
FrameMaker format	Over[Power[char[e], char[x]], num[2.0, "2"]]
Latex format	{{e}^{x}} \ over {2}

Figure 6-5: Example of converting a parse tree to a variety of formats.

Chapter 7

Results

7.1 Introduction

In this chapter the results of applying various techniques for improving decoder speed are presented, as are the effects of learning parameters in the layout model. All the tests performed in this chapter were performed on simulated character recognition data. Namely, the characters position and size data was obtained from a POSTSCRIPT file as described in Section 6.7. The are two main reasons for using simulated character recognition instead of real recognition data. First, the success or failure of mathematics recognizer can be accessed more easily if errors are not introduced from the character recognizer. Secondly, the are many details in building an accurate character recognizer that dramatically affect performance but are beyond the scope of this thesis. Appendix C contains all the randomly generated equations used in this chapter.

7.2 Definition of Accuracy

Because the correct answer when performing equation recognition is most naturally expressed as a parse tree, a notion of accuracy for parse trees must be defined. If the correct tree is T_c, then accuracy of a tree T is defined to be

$$\text{accuracy} = \frac{\text{edit-distance}(T, T_c)}{\text{number of nodes in } T_c}.$$

The edit-distance between two trees is defined in Appendix A.

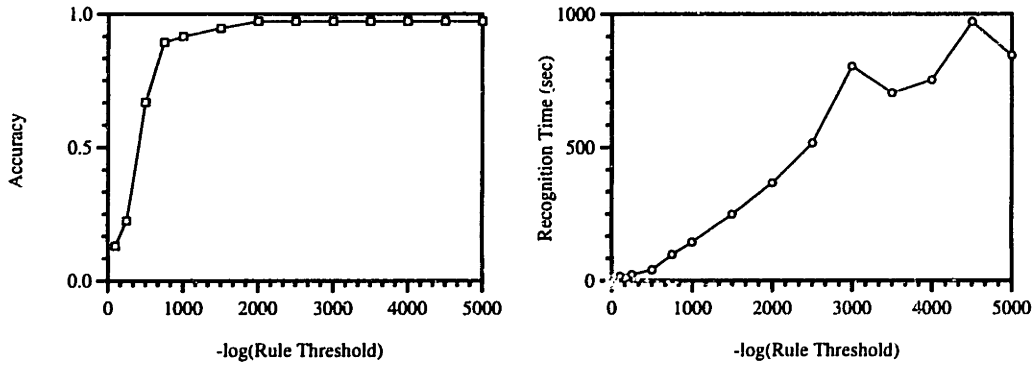


Figure 7-1: Rule Threshold versus (a) accuracy and (b) total recognition time.

7.3 Variation of the Rule Threshold

As was stated in Section 3.4.2, the rule threshold, *thres*, is the minimum allowed value for $p_E(r)$. Any node with a $p_E(r) < thres$ will be discarded. The value of the rule threshold dramatically affects the performance of the recognizer. If the value is set too high, then nodes in the valid parse tree will be discarded and the accuracy of the recognizer will suffer. If *thres* is set too low then the recognizer will waste time analyzing a large number of unlikely nodes causing recognition time to increase dramatically. Both these effects can be seen in Figure 7-1. When $-\log(thres)$ is less then 1000, the recognizer's accuracy drops off significantly. However, as $-\log(thres)$ increases past 3000, the recognition time grows rapidly without improving performance. Therefore, the optimal value for $-\log(thres)$ in this situation would be approximately 3000.

7.4 Effects of the Heuristic Estimate *est*

In Section 3.4.1 an *est* function was presented. The A^* search algorithm guarantees that as long as *est* is a conservative completion estimate (such as the one given in Section 3.4.1) then the use of *est* should not affect accuracy. Furthermore, the use of *est* should improve the recognition time. Both these predictions are confirmed in Figure 7-2. The rule threshold versus accuracy curve remains unchanged when *est* is no longer used. However, the rule threshold versus recognition time curve shifts up considerably when *est* is removed.

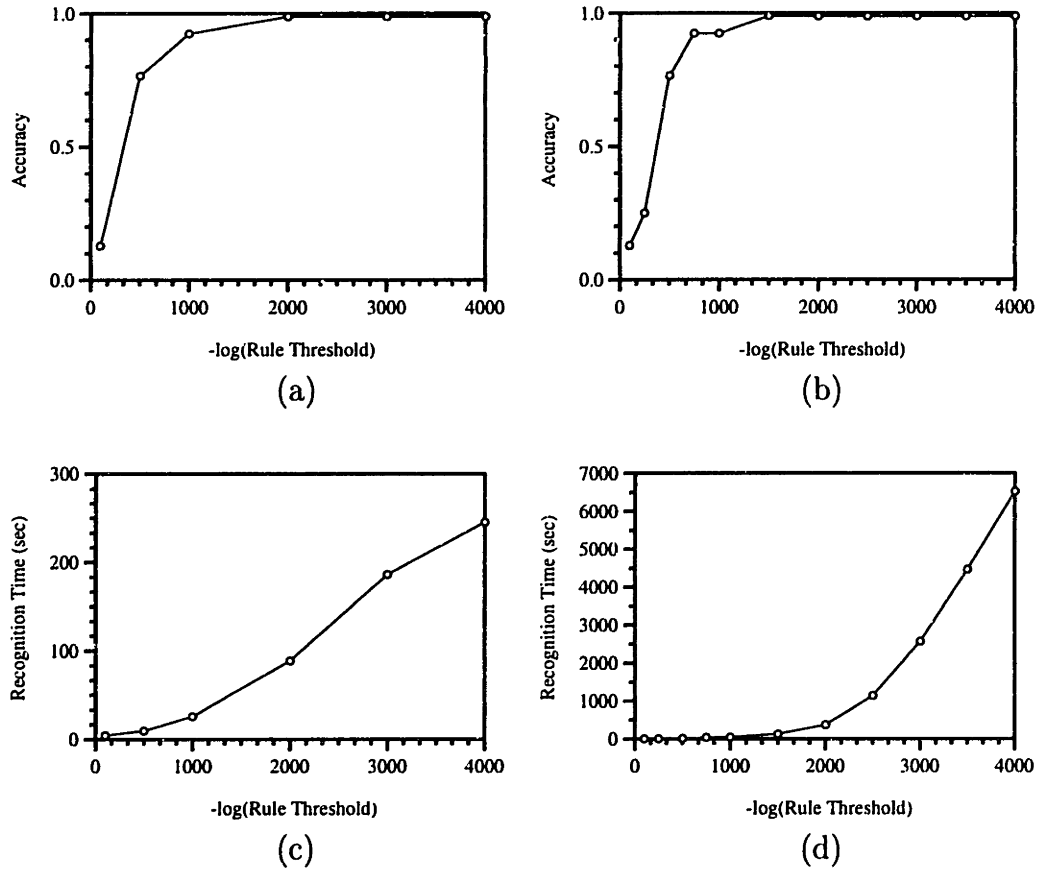


Figure 7-2: Rule Threshold versus accuracy (a) with heuristic and (b) without heuristic. Rule threshold versus time (c) with heuristic and (d) without heuristic.

7.5 Effects of the Overlap Percentage

The percentage overlap allowed between two nodes of a parse tree, like the rule threshold, must be chosen carefully for the recognizer to perform well. Just as in the case of the rule threshold, if the percent overlap allowed is set too small then accuracy will suffer, but when the percent overlap allowed is set too large the recognition time will explode without improving accuracy. Figure 7-3 confirms these predictions. In the tests in this figure the negative log of the rule threshold was set to 2000. The figure shows the overlap percentage versus accuracy and recognition time. However, because these experiments were performed using simulated character recognition data, the effect of changing the overlap percentage is not as pronounced. When using real character recognition data, characters tend to overlap considerably more because of blurring and small characters being recognized as

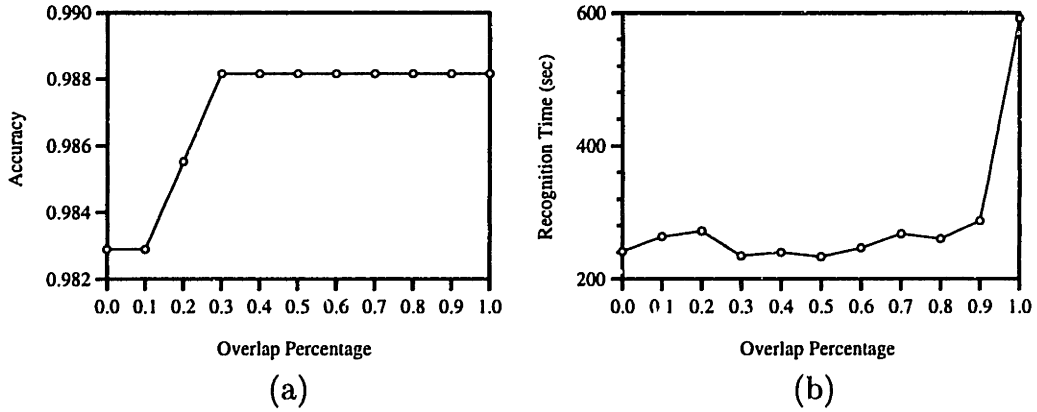


Figure 7-3: Overlap Percentage versus (a) accuracy and (b) recognition time.

pieces of larger characters.

7.6 Recognition with a Transcription

In Chapter 4 it was shown how the recognizer uses a transcription of an equation to help guide its search. Here, the prediction that recognition with a transcription is significantly easier then recognition without a transcription is confirmed. Table 7.1 shows the average accuracy and recognition time with and without a transcription for a set of 50 equations. The accuracy and speed are both improved when the transcription is used. In these test the rule threshold was set to 2000 and the overlap percentage was set to 0.3.

7.7 Learning Parameters

Until now, all the layout rules were written by hand. Namely, the coefficients in the parameterizations of T_{21}, F_0, and T_{10} were all entered manually. Here these parameters are learned automatically in a variety of scenarios. First, only T_{21} is

	Accuracy	Recognition Time (sec)
With Transcription	1.0	0.61
Without Transcription	.988	186.3

Table 7.1: The accuracy and recognition time with and without a transcription.

estimated. Then, in Section 7.7.2, the results of estimating F_0 and T_{10} are presented.

7.7.1 Results of Estimating T_{21}

Recall from Section 5 that the approach to training is to:

1. perform some initial recognition using a transcription and a "crude" layout grammar,
2. estimate parameters in the layout model,
3. perform recognition on the test data, and
4. go to Step 2 until the model stabilizes.

The "crude" layout grammar used in this section has layout functions constructed by hand for F_0 and T_{21}. However, the T_{21} function for all types of layout rules is assumed to be

$$T_{21}^{scale} = 1.0 \tag{7.1}$$

$$T_{21}^{shift_x} = F_B^{end_x} \tag{7.2}$$

$$T_{21}^{scale} = F_B^{end_y}. \tag{7.3}$$

This layout function accurately describes a multiplication rule but is a poor description of the other layout rules. Figure 7-4(b) shows an example of rendering an expression using this "crude" layout grammar. Surprisingly though, this grammar is sufficient to perform the initial recognition with a transcription.

There are two main factors that this section shall explore. First, how the size of the test sample affects accuracy, and second, how many iterations of estimation and recognition should be carried out.

As can be seen from Table 7.2, the accuracy of the recognizer improves dramatically after one iteration of training (i.e., training just on the training data). However, after the first iteration the accuracy actually decreases. This is due to the fact that the layout model, after one iteration of training, has been trained only on perfectly recognized data (the training data with transcription). However, subsequent iterations of training use imperfectly recognized data. These imperfections

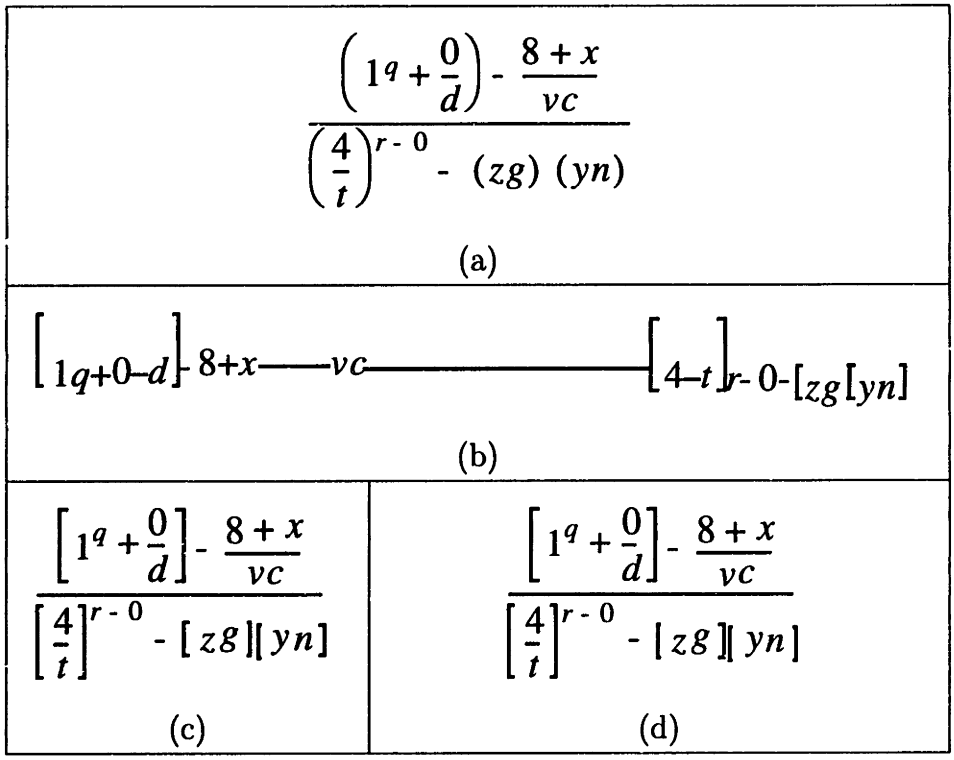


Figure 7-4: Examples of an equation laid out using various layout models: (a) original image, (b) initial layout model, (c) layout model after one iteration of training, (d) layout model after two iterations of training.

in the training data get magnified through repeated training and recognition. Figure 7-4 gives a graphical representation of the changes in the layout model through successive iterations of training.

Another reason why the accuracy does not improve as the layout model is successively retrained is that most of the information about the layout model is contained in the first 20 training examples. This idea is further confirmed by Table 7.3. In this table the number of initial training examples is increased to 50, but the over-

Layout Model	Accuracy			
	Training Data w/ transcription	Training Data wo/ transcription	Test Data	Overall
Initial	1.0	0.434	0.418	0.422
One Iteration	1.0	0.978	0.985	0.984
Two Iterations	1.0	0.973	0.978	0.977
Three Iterations	1.0	0.967	0.973	0.971

Table 7.2: Accuracy while using 20 training examples.

Layout Model	Accuracy			
	Training Data w/ transcription	Training Data wo/ transcription	Test Data	Overall
One Iteration	1.0	0.977	0.981	0.979
Two Iterations	1.0	0.970	0.976	0.973
Three Iterations	1.0	0.963	0.979	0.970

Table 7.3: Accuracy while using 50 training examples.

Layout Model	Accuracy			
	Training Data w/ transcription	Training Data wo/ transcription	Test Data	Overall
One Iteration	1.0	1.0	0.797	0.807
Two Iterations	1.0	1.0	0.925	0.929

Table 7.4: Accuracy while using 5 training examples.

all recognition rate still remains pretty much the same (it actually drops a bit). Hence, the 30 additional examples did not provide any additional information to the layout model. Finally, Table 7.4 shows the results of recognition using only 5 training examples. The recognition accuracy after one iteration of training is significantly lower than when using 20 training examples. However, the accuracy improves considerably after the second iteration of training. This is due to the fact that 5 training examples do not fully capture the layout model.

To understand why approximately 20 training examples are needed, consider the number of examples of each type of training rule. The training examples are all balanced binary trees of depth four, where each node in the tree is either addition, subtraction, multiplication, division, or exponentiation. Since each tree has 15 internal nodes, there are on average 3 examples for each of these type of nodes in each tree. Furthermore, usually 3 to 4 sets of parentheses are inserted into every tree to give the proper precedence. Therefore, each tree provides approximately 3 examples for each layout rule in the grammar. Since each T_{21} layout rule is parameterized by 7 coefficients, 20 training examples would mean about $20(3)/7 \approx 9$ equations per unknown parameter. This is approximately the minimum ratio usually needed to accurately estimate parameters. Below this, the parameters will tend to over fit the data. More equations than this tend to provide redundant information.

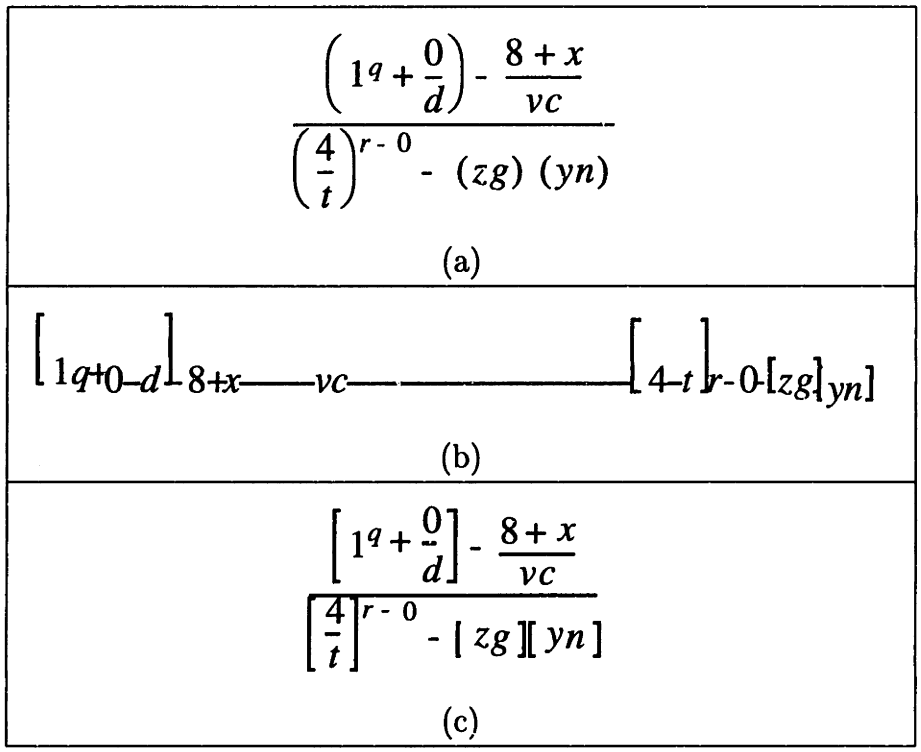


Figure 7-5: Examples of an equation laid out using various layout models: (a) original image, (b) initial layout model (including F_0 and T_{10}), (c) layout model after one iteration of training all three layout functions.

7.7.2 F_0 and T_{10}

In Sections 5.5.5 and 5.5.7 it was shown how to estimate F_0 and T_{10} in two separate steps. However, if the scale factor in T_{10} is assumed to be 1 both these functions can be estimated in one single step (i.e., they can be solved for with one linear equation). The equations for this estimation are not presented, but follow straightforwardly from those presented in Sections 5.5.5 and 5.5.7. In the results presented in this section T_{10}^{scale} is assumed to 1, and both F_0 and T_{10} are estimated simultaneously.

The initial layout model now uses

$$
\begin{aligned}
F_0^{end_x} &= F_C^{upr_x} \\
F_0^{end_y} &= F_B^{ll_y} \\
T_{10}^{shift_x} &= 0 \\
T_{10}^{shift_y} &= 0
\end{aligned}
$$

as the values for the F_0 and T_{10} functions for rules of the form $A \rightarrow BC$. The

Size of Training Data	Accuracy		
	Training Data wo/ transcription	Test Data	Overall
0	—	0.392	0.392
20	0.977	0.858	0.882
40	0.969	0.951	0.958
80	0.976	0.980	0.977

Table 7.5: Accuracy while using a varying number of training examples.

F_0 and T_{10} functions for rules of the form $A \rightarrow B$ are assumed to be provided for terminal nodes (i.e., the box metrics are known) and for nonterminal nodes they are assumed to take their trivial form. The value of T_{21} is assumed, again, to be given by (7.1)–(7.3). Figure 7-5(b) shows an equation laid out using the initial layout model.

Since all three layout functions are being estimated instead of just one, each rule's layout functions contain three times as many parameters. This means that three times the number of training examples are needed to accurately estimate the layout function. This prediction is confirmed by Table 7.5. When estimating only T_{21}, 20 training examples were sufficient but now 80 training examples are need before the accuracy improvement of giving additional training examples begins to level off.

7.8 Conclusions

From these results it is concluded that by using a few simple heuristics, recognition can be performed efficiently using the algorithm described in Section 3.4. Furthermore, an accurate model of equation layout can be constructed by using transcriptions of training data. The size of the training data must be chosen appropriately, however. In particular, when learning all three layout functions, the size of the training data must be quite large.

7.9 Future Directions

The performance improvement using the heuristic estimated presented in the thesis was rather remarkable despite the fact that it was a very conservative estimate of

the distance to completion. A much better estimate could probably be devised using the size of T_1^α and the total number of symbols to get a notion of the "distance" T_1^α is from T^α.

The layout model is probably over parameterized. Most of the layout functions are not as general as they are modeled. Learning should concentrate on either finding out which parameters are important or trying to estimate functions that the user parameterizes by hand, rather then trying figure layout from scratch. Although this work shows that the layout functions can be estimated from scratch, the ensuing layout model is not precise enough for very high quality on real character recognition data.

Because the rule threshold has such a dramatic effect on performance it should be included in the learning. The rule threshold modifies the Gaussian distribution of E by cutting off the tails. This should be handled differently for each rule and in a more rigorous manner.

Applying this work further in the handwritten domain would be interesting. Instead of trying to recognize individual characters the approach should be to only distinguish operators from characters. Furthermore, modifying the layout model for individual users could provide a significant performance advantage.

Appendix A

The Edit Distance between Two Trees

The edit distance between two parse trees is a generalization of the edit distance between two strings. For the algorithm that was used to calculate the edit distance, see [14, 6, 17]. There are three operations that can be performed on a tree: a node can be deleted, a node can be inserted, or a node can be replaced. In the discussion that follows, let a be the tree that the edit operation is being performed on and let b be the resulting tree. Furthermore, if a node y has a child x then y shall be referred to the as the parent of x and denoted $p(x)$. The edit operations are

1. A *deletion* of a node x of a, denoted $D(x)$. If a node x has children $x_1, x_2, \ldots, x_j$, and p(x) has children $y_1, y_2, \ldots, y_k$, where $y_i = x$, then in the tree b obtained by D(x), the node f(x) will have children

$$y_1, \ldots, y_{i-1}, x_1, \ldots, x_j, y_{j+1}, \ldots, y_k.$$

   Deletion of the root node of a is not valid.

2. An *insertion* of a node x into a, denoted $I(x)$. The node x will be inserted as the child of some node y of a. Node x may either be inserted with no children or else take as children any subsequence of the children of y. If in a, node y has children $y_1, y_2, \ldots, y_k$, then, for some $0 \leq j, l \leq k$ with $k \leq l$, node y in b will have the children

$$y_1, \ldots, y_j, x, y_{l+1}, \ldots, y_k,$$

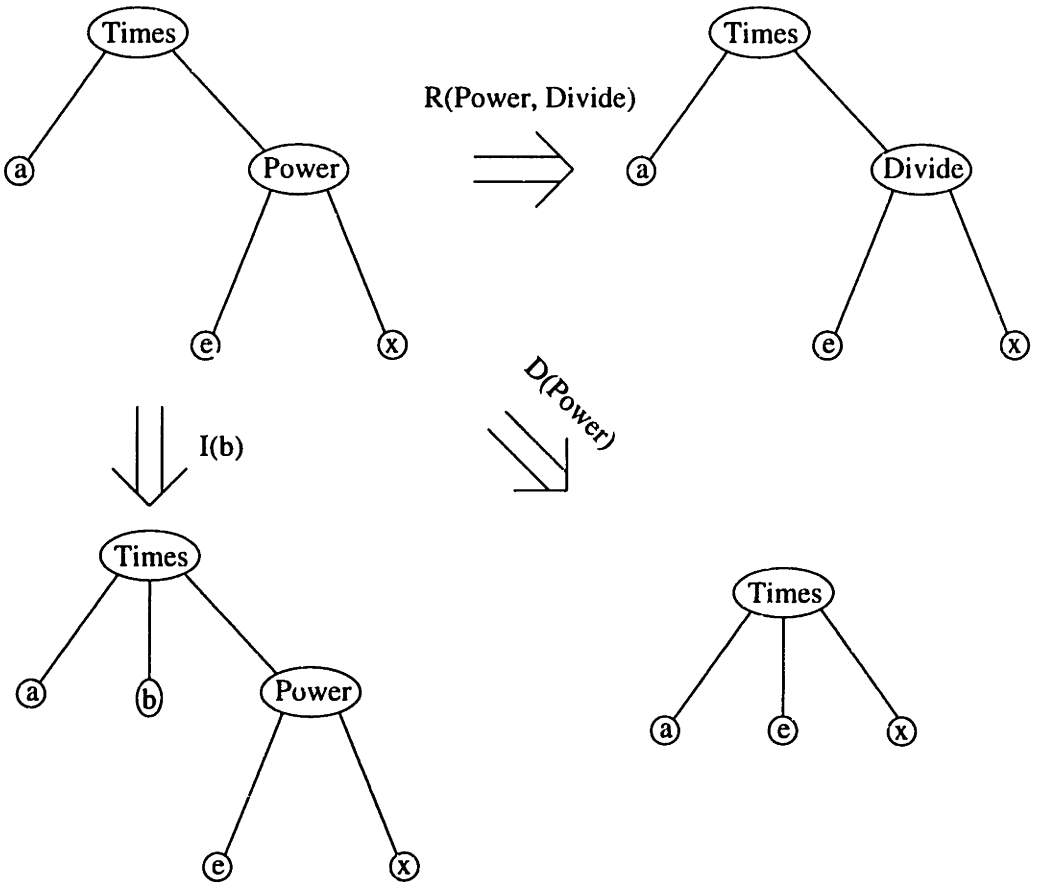


Figure A-1: Example of the three edit operations performed on ae^x.

and node x will either have no children if $j = l$, or else will have children $y_{j+1}, y_{j+2}, \ldots, y_l$. Insertion of a new root of a is not a valid operation.

3. A *replacement* of a node x of a by a node y, denoted $R(x, y)$. The node y in tree b will have the same parent and children as node x in a.

See Figure A-1 for examples of these operations. The edit distance between two trees a and b is the minimum number of the above operations needed to change a into b.

Appendix B

The Least Squares Solutions to Linear Equations

In Sections 5.5.2, 5.5.5, and 5.5.7 it was shown how the maximum likelihood estimation of various parameters was equivalent to finding the linear least squares solution $\vec{x}$ to an equation of the form

$$A\vec{x} = \vec{b} - \epsilon, \tag{B.1}$$

where A is an m by n matrix, $\vec{x}$ is an n dimensional column vector, b is a m dimensional column vector, and ϵ is an m dimensional column vector whose norm, $\epsilon^T\epsilon$, is being minimized. This Appendix shall describe the technique used to solve this equation. See [16] for a more complete treatment of this material.

Finding a solution to (B.1) is complicated by the fact that $\vec{x}$ is often both overdetermined (there are more equations then unknowns) and underdetermined (the rows of A are not linearly independent). The fact that $\vec{x}$ is overdetermined simply means that the $\epsilon^T\epsilon$ will not be 0. However, if $\vec{x}$ is also underdetermined, namely A is singular, then there is more then one solution for $\vec{x}$ that will minimize the $\epsilon^T\epsilon$. In this case one particular member of the solution set is chosen, namely the $\vec{x}$ with minimal norm $|\vec{x}|$. This $\vec{x}$ is found by using the singular value decomposition (SVD) of A. Any m by n matrix A can be decomposed into the form

$$A = UDV^T,$$

where D is an n by n diagonal matrix, and U and V are m by n and n by n matrices,

respectively. Furthermore, U and V satisfy

$$U^T U = V^T V = I.$$

Let w_i be the ith diagonal element of D. If $m = n$ and if A is non-singular, then

$$A^{-1} = V\,[\mathrm{diag}(1/w_i)]\,U^T.$$

Hence,

$$\vec{x} = A^{-1}\vec{b} = V\,[\mathrm{diag}(1/w_i)]\,U^T\vec{b}. \tag{B.2}$$

Furthermore, it turns out that when $m \neq n$ then (B.2) gives the linear least squares estimate of $\vec{x}$. Finally, if A is singular, then (B.2) is no longer meaningful because there will be at least one w_i that equals 0. However, with one slight modification it produces the $\vec{x}$ desired, namely

$$\vec{x} = V\left[\mathrm{diag}\begin{pmatrix} 0 & \text{if } w_i = 0 \\ 1/w_i & \text{otherwise} \end{pmatrix}\right] U^T\vec{b}. \tag{B.3}$$

(B.3) is the expression used to calculate the linear least squares estimate of $\vec{x}$.

Appendix C

Equations Used in Experiments

All the randomly generated equations used in the experiments in Chapter 7 are listed here. The following 10 expressions were to used generate the results in Sections 7.4 and 7.5:

$$(((t-m)-wt)+((s+t)+nk))+\left(\left(k^{9}+(f-g)\right)+\frac{3+o}{e+z}\right)$$

$$\left(\frac{g^{p}}{7^{g}}-\left(\frac{b}{k}-6^{z}\right)\right)+\left((f^{s})^{q-z}-((y+z)-py)\right)$$

$$\left(\left(\frac{7}{4}+\frac{a}{v}\right)^{y^{n}+p^{i}}\right)^{\left(\frac{m}{j}+\frac{l}{y}\right)-\left(\frac{x}{i}-(8-1)\right)}$$

$$\frac{\frac{i}{z}+du}{(h+2)^{sp}}-\left(\left(\frac{9}{i}\right)^{g+1}\right)^{(s+5)(3-8)}$$

$$\left((d^{m}+kp)+\frac{o^{1}}{b^{8}}\right)+\left(\frac{\frac{w}{x}}{j+s}+\left(\frac{j}{c}-\frac{q}{r}\right)\right)$$

$$\frac{(s-5)+\frac{r}{s}}{f^{w}-q^{f}}+(b^{s}+2^{n})^{(2z)(p-y)}$$

$$\left(\frac{7}{k}-(e-b)\right)\left(i^{b}+\frac{1}{a}\right)-\left((6b+(1-2))+(pa)\,4^{d}\right)$$

$$\left(((6-v)-(p+e))^{(9-y)(k+1)}\right)^{\left(\frac{d}{i}-\frac{w}{l}\right)-\left(\frac{4}{x}\right)^{7+c}}$$

$$\left(\frac{h}{a}+(0+9)\right)^{\frac{\frac{q}{5}}{j^{9}}}+\left(g^{1}+\frac{j}{8}\right)\left((b+g)\frac{a}{7}\right)$$

$$((k+9)\,7^{o}-((b+k)-(u-r)))-\frac{(3-8)-(m-r)}{(6\,(3))\,(9-u)}.$$

The following 50 expressions were to used generate the results in Sections 7.3 and 7.6:

$$\left(\frac{(p+m)\,(0-p)}{\frac{w}{x}}\right)\frac{(o-x)^{zw}}{\frac{v+s}{i-r}}$$

$$\left(\frac{c-9}{z+s}\right)^{v^{k}-ar}+\left((7+0)\,(i+3)-\frac{y-v}{\frac{t}{y}}\right)$$

$$\frac{v^{o}}{6-r}\left(\frac{2}{f}d^{u}\right)-((qv)\,(5+i))\,(1^{e})^{c^{9}}$$

$$\left(\frac{2-n}{x-q}\right)^{(d-u)\frac{0}{d}}+\left((l^{m}-gx)-\left(\frac{k}{1}\right)^{f+4}\right)$$

$$\left(\frac{\frac{l}{i}}{8+9}-(0d-(f+0))\right)+((c+t)-jg)\,(z+8)^{y^{f}}$$

$$\left(\left(\frac{s}{1}(7+0)\right)\left(bk+\frac{i}{7}\right)\right)^{\left((z+c)-u^{f}\right)\left((a+b)-\frac{p}{q}\right)}$$

$$(((z+8)+(u-b))-((2-z)+(p-i)))\left(\frac{q+j}{1+u}((9+i)+(o+p))\right)$$

$$\left(\frac{i^{r}}{\frac{t}{u}}-(p^{a})^{k+1}\right)\left(\frac{t-m}{c^{x}}+\frac{o-l}{3s}\right)$$

$$\left(\left(\frac{p}{q}-(s+d)\right)(k-d)^{\frac{3}{8}}\right)^{\left((g+5)j^{4}\right)^{j^{0}+(i+b)}}$$

$$\left((8-x)\frac{z}{0}\right)^{\frac{n-w}{6-z}}\left((f^{c}+mz)+\left((6-j)+\frac{1}{6}\right)\right)$$

$$\left((7c)^{q-f}-(m+7)^{\frac{l}{i}}\right)\left((in-5^{u})-\left(\frac{t}{m}+c5\right)\right)$$

$$\left(\left(\frac{y}{z}+(l+a)\right)-\frac{l2}{o+5}\right)\left((1^{2})^{8-x}-\left(\frac{c}{5}+(b+s)\right)\right)$$

$$\frac{\frac{(d^{a})^{\frac{8}{l}}}{(k+p)^{\frac{f}{o}}}}{((0-h)-z^{8})+\left(\frac{n}{s}\right)^{\frac{i}{v}}}$$

$$\left(\frac{4\left(1\right)-\left(b-0\right)}{\left(v+o\right)^{mf}}\right)^{\left(\frac{j-g}{e-r}\right)^{\frac{e}{j}\left(h-i\right)}}$$

$$\frac{\left(fo-\left(2-v\right)\right)^{2b+d^{2}}}{\left(q^{f}-\left(d-2\right)\right)^{\left(9+q\right)-0x}}$$

$$\left(\left(6-z\right)\frac{x}{6}\right)^{\frac{1^{e}}{\kappa-t}}+\left(\left(x^{u}\right)\frac{s}{5}\right)\left(o5\right)^{b4}$$

$$\left(5^{u}\left(4+p\right)\right)^{\left(w-l\right)+\frac{z}{0}}+\frac{\left(s+h\right)+\left(r-8\right)}{\left(7^{c}\right)^{q-n}}$$

$$\left(\frac{8-d}{7c}-\left(\frac{f}{w}\right)^{i+b}\right)\left(\frac{b}{c}+\left(a-v\right)\right)^{\frac{2+r}{5+u}}$$

$$\frac{\left(\left(3+s\right)+6b\right)-\left(y3\right)h^{i}}{\left(y^{z}\right)^{\frac{t}{y}}+\frac{h^{i}}{\frac{k}{h}}}$$

$$\left(\left(\frac{i}{n}-xu\right)+\left(\frac{5}{u}-s1\right)\right)+\frac{hm-\frac{4}{t}}{\left(u'+5\right)\frac{b}{o}}$$

$$\frac{\frac{t}{i}+\left(4+x\right)}{w^{l}-7w}+\left(\frac{s^{1}}{\frac{7}{4}}+\left(\left(b-8\right)-\frac{6}{b}\right)\right)$$

$$\left(\left(\left(k-l\right)+j^{c}\right)+\frac{r}{8}\left(un\right)\right)\left(\left(\left(3-k\right)-\frac{y}{b}\right)-\frac{y}{n}\left(9\left(2\right)\right)\right)$$

$$\frac{\left(\frac{b-s}{6^{7}}\right)^{\frac{a}{j}+d^{i}}}{\left(\left(i-3\right)^{x-u}\right)\frac{1-2}{4^{t}}}$$

$$\left(\frac{\left(aj\right)^{5-2}}{\frac{9(2)}{\frac{g}{1}}}\right)^{\left(\frac{5}{y}+g9\right)+\left(g^{1}\right)^{b^{s}}}$$

$$\left(t^{e}+\frac{0}{d}\right)^{\left(s-h\right)^{v-8}}+\left(\left(k+x\right)\frac{v}{0}\right)^{\left(j-w\right)-y^{v}}$$

$$\left(\frac{c^{l}}{7+w}\right)^{n^{4}2^{n}}+\left(3k\right)^{u+f}\left(m+n\right)^{1+y}$$

$$\frac{\left(\frac{j}{0}\right)^{\frac{q}{j}} + \left(\frac{e}{b}\right)^{pa}}{\frac{(6j)(x6)}{(1-e)(s+h)}}$$

$$\left(\left((e-z)+(x-2)\right)\frac{xu}{4^{l}}\right)^{\frac{d}{y}(0+p)-((c+5)+jk)}$$

$$\left(\left(5m\right)\left(8+l\right)-\frac{c^{p}}{b^{s}}\right)-\frac{\left(0-t\right)^{n^{g}}}{\frac{\frac{7}{8}}{\frac{a}{b}}}$$

$$\left(g^{t}\left(z+o\right)\right)\left(b^{g}-q3\right)-\left(\left(z^{w}-\frac{u}{z}\right)-\left(y-3\right)\frac{5}{y}\right)$$

$$\left(\frac{r+4}{u+j}-\left(u+v\right)\frac{d}{q}\right)-\left(u^{b}l^{u}\right)^{\frac{p^{i}}{o+x}}$$

$$\frac{\left(\frac{i}{n}\right)\frac{p}{2}+\left(\left(h+q\right)-\left(0-t\right)\right)}{\left(\left(p-2\right)^{k-1}\right)^{(0-p)^{r-s}}}$$

$$\frac{h^{u}}{c^{t}}\left(\left(o-9\right)+\left(v-c\right)\right)-\left(\frac{\frac{s}{5}}{ng}\right)^{(v+0)\frac{2}{b}}$$

$$\left(s^{t}\left(z-w\right)+\left(7-8\right)\frac{y}{b}\right)+\left(\left(\left(r+w\right)+6^{r}\right)-\left(a^{b}\right)^{l^{a}}\right)$$

$$\left(7^{w}\left(m+n\right)\right)^{(er)5^{q}}-\frac{u^{b}\left(l-u\right)}{\frac{t}{y}-\left(0-p\right)}$$

$$\left(\left(u^{r}\right)^{1e}\right)\frac{\frac{x}{e}}{o-1}+\left(\left(\frac{x}{a}\right)^{o-p}-\frac{c}{d}\left(j+0\right)\right)$$

$$\left(\left(\left(5+y\right)+g5\right)+\left(8p+\left(n+k\right)\right)\right)+\left(\left(o9+\frac{n}{k}\right)-\frac{\frac{j}{k}}{2^{z}}\right)$$

$$\left(\left(o-d\right)7^{s}-\left(j+0\right)\left(m-f\right)\right)^{\frac{zs}{6+j}+\left(m^{z}\right)^{l+q}}$$

$$\left(\frac{\frac{f}{0}}{\frac{e}{n}}+\left(\frac{a}{f}-\left(h-i\right)\right)\right)-\frac{y-7}{l+y}\left(\frac{d}{y}\right)^{g+h}$$

$$\frac{\left(\left(a+v\right)-\frac{x}{2}\right)-5^{a}\left(0\left(1\right)\right)}{\left(\frac{1}{y}-kd\right)\left(wd+\left(7+s\right)\right)}$$

$$\frac{\left(\left(d+m\right)0^{d}\right)\left(8+d\right)^{v-s}}{\left(\left(w5\right)^{f-g}\right)^{\frac{j-c}{2^{n}}}}$$

$$\left(\frac{4}{d}\left(b4\right)-\left(n-4\right)^{in}\right)+\frac{\frac{b}{w}+\left(e-j\right)}{y^{b}+d^{a}}$$

$$\left(\frac{n^{0}+\left(6+n\right)}{\left(u+j\right)^{x+y}}\right)^{\frac{\frac{i}{j}}{\frac{ti}{5^{m}-\left(g+t\right)}}}$$

$$\left(\frac{\frac{6}{j}}{t-u}\right)^{(t+2)+(w+5)}\left(\left(\frac{h}{u}-w^{x}\right)\frac{g+p}{b+0}\right)$$

$$\left(\frac{5}{y}\left(k+d\right)+\frac{w+1}{z+k}\right)\left(\left(\frac{w}{t}\right)\frac{f}{o}\right)^{\frac{r^{o}}{2(3)}}$$

$$\frac{0d+b4}{\frac{\frac{r}{g}}{y+3}}+\left(\left(b+o\right)+\frac{2}{v}\right)^{er+(h+y)}$$

$$\left(\left(\frac{n}{o}+\frac{6}{b}\right)-\left(\frac{u}{n}+1m\right)\right)^{\frac{(e+7)-5e}{pi+(4-t)}}$$

$$\left(i^{z}\left(5+m\right)+\left(\frac{x}{q}-wl\right)\right)+\left(\left(\frac{d}{q}-\left(0-x\right)\right)-\left(\left(s-l\right)-z^{4}\right)\right)$$

$$\left(\left(t+i\right)\left(wp\right)+\frac{0-5}{v^{c}}\right)-\left(\left(\frac{c}{9}-jw\right)+\left(3\left(8\right)\right)\left(i-v\right)\right)$$

$$\left(\frac{8^{d}\left(f-k\right)}{\left(b-o\right)\frac{m}{z}}\right)^{\left(vc-\left(i-j\right)\right)^{\frac{6-b}{\frac{9}{c}}}}.$$

The following 100 expressions were to used generate the results in Sections 7.7.1 and 7.7.2:

$$\left(\left(\frac{1}{u}-\frac{4}{l}\right)-\frac{st}{jk}\right)\left(\frac{c^{p}}{f-o}+\left(b^{k}+\left(a+3\right)\right)\right)$$

$$\frac{\left(\frac{w}{t}\right)^{jg}\left(f^{0}+u7\right)}{\frac{\frac{b}{4}}{6-z}+\left(\frac{6}{b}\right)^{9+u}}$$

$$\left(\left(f-c\right)\frac{2}{b}-\left(\frac{y}{7}-l^{i}\right)\right)\left(\left(e^{7}\left(ta\right)\right)\left(p-y\right)^{c-9}\right)$$

$$\left(\left(\left(q+j\right)-1m\right)+\left(pm\right)\left(0+9\right)\right)\left(\left(l^{e}-\left(w-t\right)\right)\frac{g-x}{f0}\right)$$

$$\left(\left(he\right)^{kl}+\left(g-p\right)^{j-s}\right)-\left(c1-7\left(0\right)\right)\left(\frac{z}{s}\left(u-b\right)\right)$$

$$(((8-9)-r^{4})-(z0+ar))+\left(\frac{b^{s}}{q+n}-\left(i^{j}+\frac{1}{q}\right)\right)$$

$$\left(\frac{b}{s}-(q-n)\right)\frac{2-b}{9-e}+\left((u+r)^{5+u}+\left(\frac{5}{a}-oh\right)\right)$$

$$\frac{\left(\frac{2-7}{5-2}\right)^{\left(\frac{9}{6}\right)^{o^{l}}}}{\frac{\frac{h}{u}+o^{9}}{(w-p)-\frac{b}{w}}}$$

$$\left((h6)\,8^{h}+\left(8^{1}\right)^{zs}\right)\left((0d)^{f-8}\left((n^{s})\,\frac{e}{b}\right)\right)$$

$$\left((k-1)\,7^{k}+\frac{j-4}{y7}\right)^{((r-g)+6z)+(iv)^{\frac{x}{m}}}$$

$$\left((r^{o}-(e+b))-\frac{a-f}{9-q}\right)+\frac{e+3}{5^{6}}\left(\frac{1}{i}+(8-p)\right)$$

$$\frac{(2-r)+5a}{\frac{dm}{k-p}}-\left(\left((9-m)-\frac{8}{t}\right)+\frac{c}{p}\,(z-0)\right)$$

$$\left((d-a)\,4^{d}-(g+1)^{\frac{i}{g}}\right)-\left(\frac{k}{x}-(z-w)\right)\left((7-c)\,\frac{i}{3}\right)$$

$$\frac{\frac{\frac{gh}{r^{3}}}{(v+s)^{\frac{y}{v}}}}{\frac{(v+8)+y^{z}}{(a-j)-9(2)}}$$

$$\left(\left(\frac{n^{g}}{6+n}\right)^{e^{3}+hi}\right)^{\frac{yj}{1y}+\frac{5}{\frac{i}{0+d}}}$$

$$\frac{((2+3)-5m)-\frac{9^{6}}{4+t}}{(p^{6}-(k+1))-\frac{c^{5}}{\frac{r}{w}}}$$

$$\frac{((xq)\,(s-h))^{\left(o^{1}\right)\frac{b}{o}}}{(0-h)\,f^{k}+((r-w)+(6+z))}$$

$$\frac{(s+x)\,\frac{7}{8}-((z-4)+(i+r))}{\left(\frac{r}{s}\,(qb)\right)\frac{6(3)}{5^{m}}}$$

$$\left(\frac{j+s}{ar}-(y^{v})\,\frac{l}{6}\right)^{((i+b)-xy)\left((h^{y})\frac{c}{1}\right)}$$

$$\left((y-z)-\frac{d}{u}\right)^{\frac{p-6}{4d}}+(xu)^{g+d}(c9)^{f+4}$$

$$\left(\left(9^{q}+(s+p)\right)\frac{ct}{3-o}\right)\frac{(wp)^{r0}}{r4+(q+v)}$$

$$\left((4+p)\frac{n}{w}+\frac{v}{4}(ab)\right)^{((7+s)+(i+j))-(i7-(t-i))}$$

$$\frac{\frac{(rs)(q-7)}{\frac{y-v}{x+a}}}{\frac{\frac{6n}{1^{2}}}{(p2)^{\frac{0}{i}}}}$$

$$\frac{\frac{y}{v}(tm)}{(d+i)-(8+t)}+((9-a)(w+t))((o+h)-fg)$$

$$\frac{(t^{6}+8h)((w-h)+(f-o))}{(o5)(nw)-(f^{s}-(y-v))}$$

$$\frac{\frac{c-p}{b-k}}{\frac{j}{0}-(u-r)}-\frac{(z-0)(y+v)}{\frac{a-3}{d+2}}$$

$$\left(\left(j^{4}\right)^{y+z}\right)\frac{6+b}{x2}-\left(\left(\frac{6}{7}\right)^{\frac{5}{i}}-((l-q)-(o-h))\right)$$

$$\frac{\frac{e-f}{p+m}}{\left(\left(\frac{t}{i}\right)^{c^{h}}\right)^{(8-1)-(j-4)}}$$

$$((t+i)(w1))\left(ot+\frac{n}{0}\right)-(((4+d)-zk)+((n+4)+(i+n)))$$

$$\left(\left(\frac{g}{t}+\frac{7}{o}\right)+(v+k)^{a-j}\right)-\left(\left(\frac{z}{o}+yj\right)+(a+3)(py)\right)$$

$$(((3+s)+(q+3))+(ij-x2))^{\frac{e7}{\frac{d}{y}}-((d+6)+(w+d))}$$

$$\left(\left((m+v)^{l^{6}}\right)^{\frac{x}{\frac{1}{wh}}}\right)^{\left(p^{i}\right)^{c^{h}}((k+h)+(b-8))}$$

$$\frac{\left(\frac{l}{y}\right)^{4^{9}}}{\frac{wl}{\frac{7}{0}}}((s-t)+7(0))^{(vw)^{a-b}}$$

$$\left(\left(\frac{\frac{g}{x}}{b^{g}}\right)^{\frac{7-k}{if}}\right)^{\frac{L+c}{2^{z}}+\left(e^{n}-\frac{h}{a}\right)}$$

$$\frac{(j-o)+a3}{y^{3}\left(ty\right)}-\left(\left(2-r\right)^{hq}+\left(\left(p+2\right)+\left(c+t\right)\right)\right)$$

$$\frac{\left(\frac{q}{z}+\left(h-q\right)\right)-5^{2}w^{t}}{\left(\left(t+q\right)-o^{d}\right)\left(0^{p}+n8\right)}$$

$$\left(\left(\frac{d}{i}\right)^{c-1}\right)^{(wh)(j-w)}+\frac{8^{h}+\frac{n}{g}}{\frac{b^{o}}{y-f}}$$

$$\frac{\left(s-5\right)^{\frac{v}{s}}-\left(7w-y^{3}\right)}{\frac{rg-(q-7)}{\frac{2}{b}-hi}}$$

$$\left(\left(3\left(4\right)+q7\right)-\left(ij-\left(9+y\right)\right)\right)+\left(\left(\left(a-z\right)-de\right)+\left(p^{u}\right)^{k-5}\right)$$

$$\left(\frac{6b}{5-e}\left(\left(l-2\right)-gx\right)\right)\left(\left(ty\right)\left(o-h\right)\right)^{\frac{k+5}{\frac{L}{c}}}$$

$$\left(h^{y}+c^{l}\right)\frac{0+5}{3-g}-\left(gx\right)^{b-0}\left(b^{c}\right)^{m+r}$$

$$\left(4^{h}v^{g}+\frac{j^{8}}{af}\right)+\left(\frac{z^{g}}{q^{j}}\right)^{\frac{6+7}{pq}}$$

$$\left(\left(\left(n^{o}\right)^{\frac{6}{v}}\right)^{\frac{m}{n}-tq}\right)^{\frac{\frac{m^{\frac{b}{k}}}{\frac{h}{2}}}{(1+m)^{w-d}}}$$

$$\left(\left(\left(ez\right)^{\frac{l}{m}}\right)\frac{9u}{\frac{k}{h}}\right)^{\left(p^{a}-(w+d)\right)^{(g-5)^{v+s}}}$$

$$\frac{\left(1^{q}+\frac{0}{d}\right)-\frac{8+x}{vc}}{\left(\frac{4}{l}\right)^{r-0}-\left(zg\right)\left(yn\right)}$$

$$\left(\frac{\left(s+1\right)+\left(z+g\right)}{\left(r+s\right)-i^{v}}\right)^{\frac{\left(\frac{y}{o}\right)\frac{y}{j}}{\frac{y}{j}+\frac{t}{u}}}$$

$$\left(\left(fc-a7\right)+\left(q+r\right)\frac{h}{2}\right)^{(m7+de)-(1a-wl)}$$

$$\left((6-n)\left(9+y\right)-\left(ti+\left(k+d\right)\right)\right)^{\frac{x+2}{\frac{0}{d}}(0x-(v+w))}$$

$$\frac{\left(\left(x+a\right)+\left(g+9\right)\right)^{\frac{4-t}{z4}}}{\frac{4(5)}{v+8}-\frac{n+0}{q^{7}}}$$

$$\left(w^{9}\right)^{3+w}\left(\frac{b}{o}-a^{v}\right)-\left(n-k\right)^{an}\left(\left(u+f\right)-\frac{p}{a}\right)$$

$$\frac{f0+\left(2+7\right)}{av+p2}-\left(\left(q^{3}-lm\right)+\left(\left(x-q\right)-s1\right)\right)$$

$$\frac{\frac{(6+j)-\frac{1}{y}}{l^{i}-\frac{8}{1}}}{\frac{xu}{s^{l}}+\frac{k^{p}}{j^{s}}}$$

$$\left(\left(\frac{5}{a}\right)^{4^{9}}-\left(\left(g-9\right)+\frac{3}{4}\right)\right)-\frac{\left(4-t\right)^{3-0}}{\left(f-o\right)+\left(q+z\right)}$$

$$\left(\frac{\frac{4}{x}}{r0}+\left(zs+\left(i-b\right)\right)\right)\left(\left(z^{g}\right)^{6b}\right)^{y^{r}d^{6}}$$

$$\frac{\left(\left(b^{c}\right)^{ef}\right)^{(uv)5^{m}}}{\left(q7\right)^{1-u}-\left(t^{e}+wx\right)}$$

$$\left(\left(q-f\right)^{5-q}-\left(h+6\right)^{\frac{k}{i}}\right)-\left(\left(5i\right)^{\frac{k}{l}}-\left(o^{5}+\frac{b}{s}\right)\right)$$

$$\left(\left(d^{e}c^{5}\right)\left(4h\right)^{r+k}\right)\left(\left(\left(g+1\right)-ns\right)\left(\left(b+c\right)\frac{6}{f}\right)\right)$$

$$\frac{\frac{ct}{b^{k}}\left(\left(b+8\right)\left(y+r\right)\right)}{\left(nc+a^{v}\right)^{\frac{2}{7}-(d+e)}}$$

$$\left(\frac{z^{s}}{y^{3}}-\left(2n\right)\frac{1}{y}\right)\frac{\left(\frac{m}{j}\right)^{hq}}{\left(p-y\right)^{\frac{c}{i}}}$$

$$\frac{\left(2^{n}+i^{2}\right)-\left(\left(t+y\right)+\left(s+5\right)\right)}{\frac{\frac{\Omega}{q}}{o^{p}}+\left(wd\right)^{n-8}}$$

$$\left(\left(5m\right)\left(w+l\right)\right)^{\frac{o+d}{\frac{t}{c}}}\left(\left(k1\right)\left(r-8\right)-f^{c}\left(av\right)\right)$$

$$\left(\left(\left(\frac{c}{p}+\left(b+4\right)\right)\left(\frac{r}{k}\left(yz\right)\right)\right)^{((v-0)(av))^{\left(2^{b}\right)^{5^{y}}}}\right)$$

$$\left((f8)^{uj} - (a^{n} + xq)\right)\left(\left(\frac{m}{7} + p^{q}\right) + \left(9\,(6) - \frac{w}{l}\right)\right)$$

$$\frac{(m^{n} + d^{y})^{(1-i)+g^{5}}}{\frac{5}{m}\,(s+t) - ((0-1) - z^{o})}$$

$$\left(((9+a) + (4+x))^{\frac{4}{5}-(n-g)}\right)^{\frac{kx}{3^{w}}(z^{c})^{2^{7}}}$$

$$\left((w+x)^{r-c} + \frac{j8}{\frac{6}{r}}\right)\left((r-o)^{i-v}\right)^{\frac{q+r}{19}}$$

$$\left((7^{s})^{a^{z}} + (2v)^{l+2}\right)((i-f) - tq)^{(p-e)-(0+l)}$$

$$\frac{\left(\frac{i}{v}\right)^{\frac{h}{m}} - (hi)\,\frac{k}{d}}{(9+a)^{g+p}\,((c+p) - 7k)}$$

$$(pu + 8p)\,\frac{g+p}{f+0} - \left(c^{d} + (n+g)\right)\frac{f^{c}}{y+3}$$

$$\left(\left((8-9) - \frac{r}{4}\right) + \left(\frac{3}{0}\right)^{a^{r}}\right) + ((3k - 6z) - ((u+7) - 5^{q}))$$

$$\left(\left((7+0) - \frac{2}{n}\right) - (u+v)^{h-u}\right) - \left(\frac{y3}{l-m} + \frac{1-y}{\frac{c}{h}}\right)$$

$$\frac{(a^{j})^{\frac{9}{a}}}{\frac{l}{m} - (g-t)} - \left(\frac{h^{y}}{0-5} + \frac{s^{1}}{3^{8}}\right)$$

$$\left(((1-6) - 8^{d}) - \frac{4^{t}}{n0}\right)\left((gd)\,\frac{f}{w} + \frac{b-8}{6^{v}}\right)$$

$$\frac{(gh)^{3+c}}{(f+o) - 6f}\left(\frac{7k}{\frac{i}{3}}\right)^{2z+9^{y}}$$

$$(((r+w) - (2-r))\,(2b - (x+u)))\left((qj + (h-y)) - \left(d6 + \frac{s}{h}\right)\right)$$

$$\left(\frac{y}{n}\,(5-m) - \left(\frac{5}{a} - c9\right)\right) + \left(\frac{p}{u}\,(k-d) + (cx)^{7+g}\right)$$

$$\left((h+m) - 4^{1}\right)^{\frac{s}{h}-(z+k)} + (4t)^{7+g}\left(7s + \frac{u}{f}\right)$$

$$\frac{((4+d) + (7-g)) - \frac{r+4}{a^{v}}}{\left(\frac{\frac{r}{0}}{q+f}\right)^{\frac{\frac{y}{f}}{2}}}$$

$$\frac{\left(\frac{3}{c}\right)^{a^n}}{\frac{ab}{p^u}} + \left((2+r)+xi\right)^{\left(l^2\right)^{k-p}}$$

$$\left(\frac{\frac{2}{v}}{1+q} - \frac{9-y}{8-1}\right)^{\frac{\frac{p}{q}}{s+d}+((w+h)-(n-o))}$$

$$\frac{\left(p^q\right)^{\frac{8}{x}}}{(4-5)\,\frac{f}{4}}\left((c-h)^{v8}\right)^{\left(\frac{7}{0}\right)^{i+f}}$$

$$\left((c-9)-\frac{r}{o}\right)\left(z^o-\frac{6}{f}\right)+\left(\left(j^w\right)^{\frac{a}{j}}-\frac{e^n}{\frac{d}{e}}\right)$$

$$\left(\frac{\frac{z}{o}}{\frac{2}{3}}\left(\frac{2}{n}\right)^{5m}\right)\left(\left((y+j)-(1-u)\right)\left(x^m+4^9\right)\right)$$

$$\frac{\left(e^v+\frac{1}{i}\right)-(x6)\,(g-5)}{\left(\frac{h^i}{w5}\right)^{(4-p)n^8}}$$

$$\frac{\left(h^i\,(oh)\right)^{\frac{k}{l}-\frac{v}{8}}}{\left(\frac{\frac{c}{p}}{7-4}\right)^{j^k-2^3}}$$

$$\left(\left(\frac{w}{x}+(n+o)\right)\left((n+8)+(a-7)\right)\right)^{\left(\frac{v}{c}\right)^{ur}((2+b)(p-6))}$$

$$\left(\left((7-o)+\frac{2}{r}\right)+\left(\frac{6}{z}+dm\right)\right)^{\left(\frac{\frac{i}{n}}{\frac{5}{m}}\right)\frac{p}{\frac{c}{g5}}}$$

$$\left((e-3)^{\frac{d}{u}}+\frac{h^a}{s+x}\right)\left(\left(di-\frac{0}{h}\right)\frac{w^9}{b^g}\right)$$

$$\frac{(le)^{sd}}{\frac{o+t}{fk}}-\left(\left((w+h)+\frac{b}{4}\right)-\frac{f4}{2-3}\right)$$

$$\left(\frac{kd}{7-s}-\left(j^c\right)^{\frac{q}{7}}\right)-\frac{\frac{z}{c}}{\frac{i}{b}}\left(qz+\frac{9}{u}\right)$$

$$\left(\frac{v4}{m+7}+(uv)^{t-q}\right)^{(u-3)(t-2)-((p-y)-4(5))}$$

$$\left(\frac{6+j}{h-m}-\frac{d+y}{g-1}\right)-\left(\left(pa-\frac{0}{9}\right)-\frac{k-9}{j-8}\right)$$

$$\frac{\frac{5e+(w+h)}{(8-d)-\frac{7}{w}}}{\left((4^{p})\,\frac{3}{o}\right)^{\left(\frac{v}{g}\right)\frac{i}{v}}}$$

$$\left(\frac{k-1}{7+k}+\left(\frac{7}{w}+(u-f)\right)\right)^{\left((f+w)-e^{j}\right)-(2z)^{\frac{l}{2}}}$$

$$\frac{\frac{\frac{f}{k}}{\frac{y}{n}}-(u-j)^{pu}}{\frac{\frac{y}{3}(t+6)}{(h-u)+(g+t)}}$$

$$\frac{(6-z)\,(pe)}{\frac{x}{q}+(g-5)}-((t+y)\,(k-p)-(4\,(9)-(7+k)))$$

$$\frac{\left((9e)^{8+d}\right)\frac{o-d}{7+4}}{((k+t)-(b+k))-((v+g)-mf)}$$

$$\frac{\frac{\frac{c}{l}\,(bw)}{\frac{7}{0}-qr}}{\left(\frac{7}{g}\right)^{m-b}+((2+f)+h^{2})}$$

$$\left(\left(\frac{r}{o}+qr\right)+(u+z)\,(ti)\right)^{\frac{\frac{a}{7}}{\frac{1}{6}}-\frac{d^{y}}{\frac{c}{p}}}$$

$$\frac{\frac{(y-3)-\frac{d}{e}}{(9-m)+(c+p)}}{(d^{6})\,\frac{s}{1}-o^{5}j^{s}}.$$

Bibliography

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley, Reading, MA, 1986.

[2] A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation, and Compiling*, volume I: Parsing. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[3] Dennis S. Arnon and Philip A. Chou. Blending structural and image-level editing of documents. Preprint.

[4] Steven C. Bagley and Gary Kopec. Editing images of text. *Communications of the ACM*, 37(12):63–72, December 1994.

[5] P. A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In *Visual Communications and Image Processing*, pages 852–863, Philadelphia, PA, November 1989. SPIE.

[6] Kuo chung Tai. The tree-to-tree correction problem. *Journal of the Association for Computing Machinery*, 26(3):422–433, July 1979.

[7] K. S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Englewood Cliffs, NJ, 1982.

[8] R. C. Gonzalez and M. G. Thomason. *Syntactic Pattern Recognition: An Introduction.* Addison-Wesley, Reading, MA, 1978.

[9] M. A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, Reading, MA, 1979.

[10] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[11] G. E. Kopec and P. A. Chou. Document image decoding using Markov source models. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 16(6):602–617, June 1994.

[12] Gary Kopec and Maurico Lomelin. Document image decoding approach to character template estimation. *IEEE Transactions Pattern Analysis and Machine Intelligence*, November 1995. Submitted.

[13] S. E. Levinson. Structual methods in automatic speech recognition. *Proceedings of the IEEE*, 73(11):1625–1650, November 1985.

[14] Andrew S. Noetzel and Stanley M. Selkow. An analysis of the general tree-editing problem. In David Sankoff and Joseph B. Kriskal, editors, *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*, chapter 8, pages 237–252. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1983.

[15] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, 1984.

[16] William H. Press, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, New York, New York, 1988.

[17] Stanley M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6(6):184–186, June 1977.

[18] K. Sam Shanmugan and Arthur M. Breipohl. *Random Signals.* John Wiley & Sons, New York, New York, 1988.

[19] Tichomir Tenev. Correspondence.

[20] Patrick Henry Winston. *Artificial Intelligence.* Adison-Welsley Publishing Company, Reading, Massachusetts, 1992.