The Activity Metric for Low Resource, On-Line Character Recognition

Except where reference is made to the work of others, the work described in this dissertation is my own or was done in collaboration with my advisory committee. This dissertation does not include proprietary or classified information.

William James Confer

Certificate of Approval:

W. Homer Carlisle
Associate Professor
Department of Computer Science and Software Engineering

Richard Chapman, Chair
Associate Professor
Department of Computer Science and Software Engineering

Dean Hendrix
Associate Professor
Department of Computer Science and Software Engineering

Stephen L. McFarland
Acting Dean
Graduate School

The Activity Metric for Low Resource, On-Line Character Recognition

William James Confer

A Dissertation

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Doctor of Philosophy

Auburn, Alabama
16 December 2005

The Activity Metric for Low Resource, On-Line Character Recognition

William James Confer

Permission is granted to Auburn University to make copies of this dissertation at its discretion, upon the request of individuals or institutions and at their expense. The author reserves all publication rights.

Signature of Author

16 December 2005

Date of Graduation

Vita

William Confer began his career in the field of Computer Science early, being exposed to programming in the early 1980's at home and at the Classical Junior Academy of St. Louis, Missouri. Upon completing the Computer Science program at Illinois College in 1999, William worked as a software developer for the Department of Veteran Affairs, Veteran Hospital Division and then moved south to Auburn, Alabama where he began his graduate career. While at Auburn University, William has worked hard in the fields of character recognition and wireless software development. His efforts in character recognition have culminated in this doctoral work and a U.S. patent he shares with his advisor, Richard Chapman.

Dissertation Abstract

The Activity Metric for Low Resource, On-Line Character Recognition

William James Confer

Doctor of Philosophy, 16 December 2005
(M.S., Auburn University, 2005)
(B.A., Illinois College, 1999)

196 Typed Pages

Directed by Richard Chapman

This work presents an algorithm for on-line character recognition that is fast, portable, and consumes very little memory for code or data. The algorithm is alphabet-independent, and does not require training beyond entering the alphabet once. This algorithm uses a novel, parameter-based method of feature extraction, *activity*, to achieve high recognition accuracy. Recognition accuracy is shown to be improvable dynamically without further input from the user. The algorithm brings the capability to do character recognition to classes of devices that heretofore have not possessed that capability because of limited computing resources, including mobile handsets, PDAs, pagers, toys, and other small devices. It achieves recognition speeds of 16.8 characters per second on a 20MHz, 8-bit microcontroller without floating-point. The alphabet-independent nature of the algorithm combined with its inherent resistance to regular noise interference may allow it to enhance the capability of persons with impaired motor or nervous systems to communicate with devices by writing or gesturing commands. Additionally, two human studies demonstrate the effectiveness of a simple, activity-based recognizer for users of the stylized Graffiti alphabet and for non-stylized variants of the English alphabet. A final experiment shows how recognition accuracy can be improved per user by modifying the parameters of the activity metric over samples collected in the non-stylized study.

Acknowledgments

This work is dedicated to the memories of my grandmother, Lois Hoffman, and my truest friend, Jacob Palmatier. Lois passed recently after a twenty-plus year battle with multiple sclerosis. She is the inspiration behind the majority of my work with character recognition. Jacob was killed by a roadside bomb in Iraq while retrieving the mail. He was my only friend who didn't criticize pursuing higher degrees for all these years. I miss each of them terribly.

I would like to thank each of my committee members, especially Richard Chapman, and my outside reader, Chwan-Hwa Wu, for their support over the years. Jaun Gilbert supplied the tablet computers used to collect samples for my final studies. Gerry Dozier's assistance in evolutionary techniques was crucial to the success of my attempts to optimize recognition. Mike Spiegel, an undergraduate student from Depauw University, was a tremendous help in organizing and collecting character samples. I would also like to acknowledge the contributions of the following students for their assistance in the past: Charlton Barker, Tyson Begly, David Boyette, Barry Burton, Crystal Collings, Jim Han, Travia Holder, Kevin Jackson, Justin Limbaugh, Adam Luter, Deitrick Mathews, John Morley, Christopher Nuby, Marcus Parker, and Bradley Scott. An additional round of thanks go to Auburn University Technology Transfer for assistance in acquiring the patent and in seeking licensees for activity-based recognizers.

This work was sponsored in part by the Auburn University Center for Innovations in Mobile, Pervasive, Agile Computing Technologies (IMPACT) and the U.S. Department of Education Graduate Assistance in Areas of National Need (GAANN) Fellowship.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliography follows van Leunen's *A Handbook for Scholars.*

Computer software used The document preparation package TeX (specifically LaTeX) together with the departmental style-file `aums.sty`.

TABLE OF CONTENTS

List of Tables

List of Figures

Chapter 1
Introduction

Since the mid 1990's on-line character recognition has become widely employed in Personal Digital Assistants (PDAs), beginning with the Palm Pilot devices which defined the product category. However, a number of factors have limited the use of character recognition to this category of device, and has even, for some PDA users, proved too frustrating. These include lower real-world accuracy rates than advertised, fairly significant requirements for memory and processor speed, and dependence on a stylized alphabet that users are forced to learn.

This work presents an algorithm that, by means of a novel feature extraction technique, *activity*, significantly reduces the computational overhead required to support robust, on-line character recognition and permits the use of arbitrary alphabets. There are quite a few applications for such an algorithm. First, devices with very little computational capability can now incorporate character recognition. Four implementations of the algorithm will be described — one on a 20MHz, 8-bit microcontroller using 40K bytes of memory. Thus, toys, pagers, mobile phones, and many other small, cheap devices can take advantage of character recognition for command and data entry. Second, the alphabet independence of the algorithm makes it attractive for use by those who require application specific alphabets or gestures. Any set of marks can be assigned arbitrary meanings since the algorithm doesn't use particular features of the Roman alphabet or any other. The parameters of the algorithm are suitable for modification (post-deployment) so that the idiosyncrasies of the writing of any particular user can be incorporated and thus improve recognition accuracy.

Finally, this algorithm, in practice, appears to exhibit an immunity to noise that makes it forgiving of the writing style of someone writing in a noisy environment (such as on a subway or bus, for example), or suffering from a tremor, nervous or motor condition.

Three studies will investigate the real world performance of simple, activity-based recognizers. The first allowed users to interact with the recognizer using the stylized Graffiti alphabet shipped on devices running the Palm OS, pen-based operating system. Since this alphabet is supported by a large number of existing mobile platforms (including non-Palm OS systems), the results of the study afford insight into the immediate usefullness of the activity metric for contemporary market devices. The results of this study measured average recognition accuracy from 97.12% (for experts) to 95.01% (for novices). A second study allowed subjects to perform their non-stylized version of characters in the English alphabet in a non-interactive setting – i.e., the recognizer was applied to the captured drawings in an off-line fashion, once the user has completed their writing. This affords data that is void of character adaptions a human might traditionally make to comprise for deficiencies in a recognizer (eg, exaggerating the bumps of a 'B'). Additionally, both the upper and lower case alphabets are examined whereas there is only one case in Graffiti. The results of this second study for alphabets with three samples of each letter measured an average upper case accuracy of 92.2% where 83% of subjects' accuracy exceeded 90% and an average lower case accuracy of 91.6% where 71% of subjects' accuracy exceeded 90%. The third study examines the effects of altering the parameters of the activity metric for each user's data from the previous study. The focus of this final study is on improving the recognition accuracy for each user. This final experiment resulted in an average upper case accuracy of 94.3% where 92% of subjects' accuracy exceeded 90% and an average lower case accuracy of

93.1% where 86% of subjects' accuracy exceeded 90%. Further, the error rate was reduced by an average of 30.3% for upper case and 20.9% for lower case characters.

Chapter 2
Related Work

Character recognition is a mature field. Interfaces using handwritten/gestural input were being researched as early as the late 1950s [7, 13, 16] including the use of photo-sensors for the recognition of hand gestures such as waving goodbye. Early methods are surveyed in [14, 27, 45, 47]. The development of high resolution optical scanners and digitizing tablets during the 1960's and early 1970's, fueled both character and handwriting recognition. Most methods were off-line processing methods which used bitmap and string contextual information to increase recognition accuracy. The algorithm presented in this work is an on-line method.

2.1 On-Line vs. Off-Line Recognition

Character (or more generally, pattern) recognition systems are classified as either on-line or off-line systems, dependant on the way drawings of characters are analyzed.On-line systems often (although not always) interact with users during the drawing process. They take advantage of temporal data gathered from the pen events of a stylus and pad, for example, in order to analyze how the drawing was put down by the user. This data can be as simple as sequenced X and Y coordinates, but may also include additional information such as pressure, timing, and stylus angles [20, 47]. Generally, on-line methods sacrifice accuracy for real time performance speeds, which is counter to off-line recognition [20, 27, 47]. There is some debate as to whether temporal data merely adds noise to the static images produced

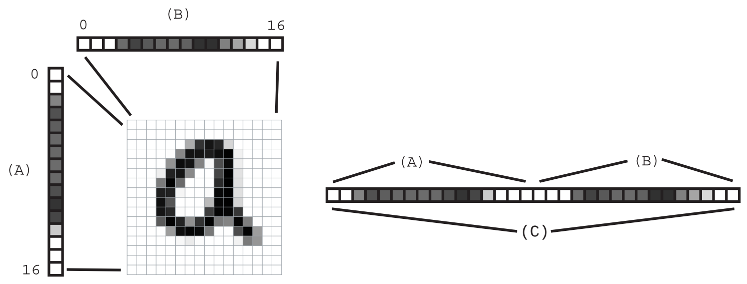


Figure 2.1: (A) Vertical histogram of the letter ‘a’. (B) Horizontal histogram of the letter ‘a’. (C) Compound histogram of the letter ‘a’.

by the stylus; however, Kassel [27] has demonstrated the additional information can be used effectively to increase recognition accuracy in a variety of systems.

Off-line methods evaluate the pixel information obtained by optically scanning an existing document, for example. Optical Character Recognition (OCR) is a commonly used form of off-line recognition. One simple approach used in off-line systems is to normalize the size of character drawings and evaluate the one-dimensional projection of the resulting bitmaps. Figure 2.1 shows example vertical, horizontal, and compound ((A), (B), and (C) respectively) histograms of the letter ‘a’.

Further coverage of off-line methods is beyond the scope of this research; however, there are several thorough surveys on the subject [28, 39, 44]. Koerich et al survey large scale recognition systems that combine core character classifiers (“what letter is this?”) with vocabulary contexting (“given the last few letters and some dictionary, what letter is this?”) [28]. That survey focuses particularly on systems with very large dictionaries – some tens of thousands of words – such as the recognition system used by Microsoft Tablet PCs. Like Koerich, Steinherz et al discuss the methods of systems that include word recognition; however, the systems evaluated are all cursive script readers [44]. Plamondon et al cover the fundamental techniques of both on-line and off-line recognition specifically as they apply to the fields of signature verification, writer authentication, and handwriting learning [39].

Presently, much research in on-line character recognition has centered around single character entry systems [5, 3, 4, 10, 11, 12, 19, 20, 22, 23, 27, 31, 25, 24, 33, 34, 48, 49]. Characters are entered one at a time and the recognizer classifies the character before the next is written. This provides the user immediate feedback so that errors can be corrected as they occur. Typically, there is a simple method for the user to depict the beginning and end of each character - commonly accomplished by pen down and up events.

2.2 Unistrokes

Unistrokes [19], developed at Xerox Corporation in 1993 is a well known example of a single character, pen-event system. Unistrokes characters were designed to be written one on top another so as to minimize the real estate required for recognition and to allow for "eyes free operation" [19]. The Unistrokes alphabet is based on five basic strokes and their rotational deformations. While several characters ('i', 'j', 'L', 'O', 'S', 'V' and 'Z' for example) are represented by strokes similar to their Roman drawings (see Figure 2.2), most characters' strokes require unnatural memorization [33]. Additionally, a model has been developed for predicting the time required to enter arbitrary text with Unistrokes by an expert user [22]. This is particularly useful since several variations of the Unistrokes alphabet have been introduced in recent years [22].

A popular variant of Unistrokes is the *Graffiti* system originally used in the Palm OS family of PDAs [1]. Graffiti improved upon Unistrokes by representing characters with symbols that are, for the most part, quite like their Roman counterparts (see Figure 2.3).

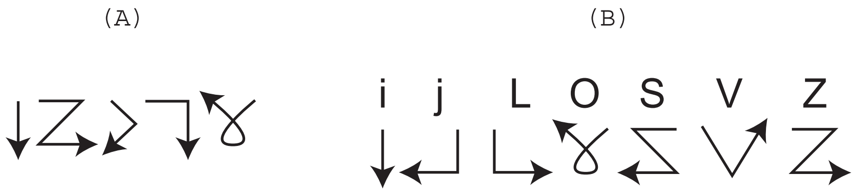


Figure 2.2: (A) Five strokes of the Unistroke alphabet. (B) Unistroke letters that map directly to their Roman letters.

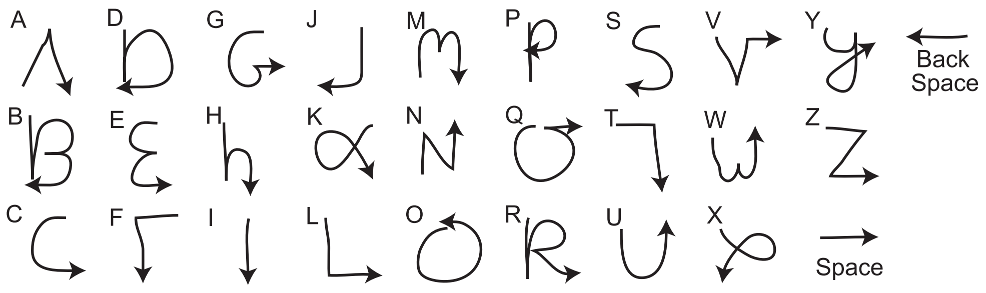


Figure 2.3: The Graffiti alphabet

A disadvantage of both Graffiti and Unistrokes is that their alphabets are static. Graffiti also has several characters that are composed of multiple strokes in order to allow a more natural writing style. As users change applications, more or fewer characters may be required [12, 11]. For example, there is little need for a simple, arithmetic calculator to recognize characters other than say digits, some punctuation and operators. Reducing the size of the alphabet in these situations might also increase recognition accuracy.

2.3 Self-Disclosing Systems

T-Cube [48], developed at Apple Computers in 1994, is a self-disclosing method for character input. Nine pie-shaped templates designate the alphabet map as in Figure 2.4(A), each pie cut into eight wedges. Each wedge contains characters or character commands. . . Figure 2.4(A)

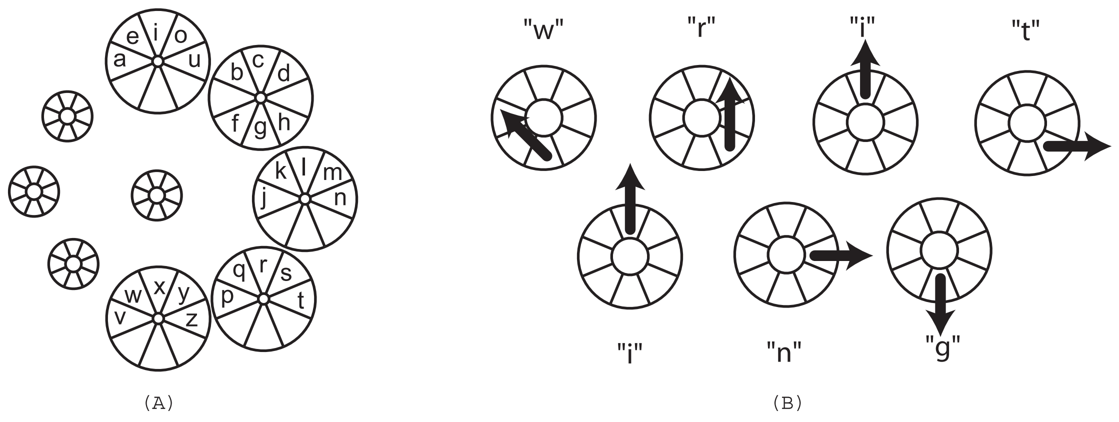
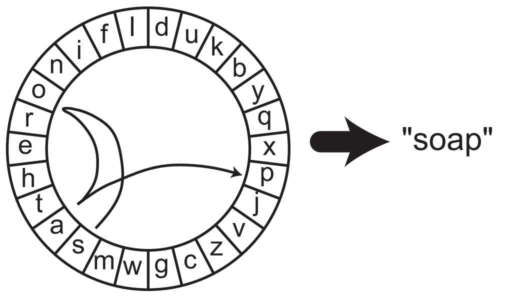


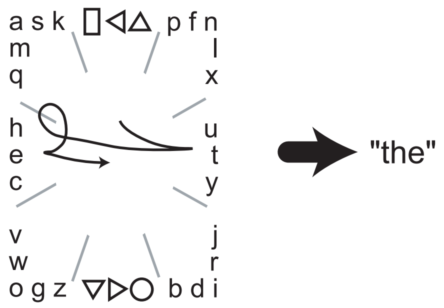
Figure 2.4: (A) The alpha character layout of T-Cube. (B) T-Cube flick sequence for the word ”writing”.

only demonstrates the location of the alpha characters for simplicity. Characters could be input essentially by touching a stylus to the desired wedges in sequence. To reduce the use of precious screen real estate, however, the T-Cube user only draws on a single pie target like those shown in Figure 2.4(B). This target has an enlarged center, giving the pie nine wedges. The user is able to perform any of the characters from the expanded map by “flicking” a stylus from the center of a wedge in any of the eight cardinal directions. The wedge pen down event represents which of the nine pies in the map the character is to be recognized from. The direction of the flick determines which wedge of this pie to recognize. This approach significantly decreases the amount of stylus-to-pad time required to draw an arbitrary character since each drawing is a unidirectional flick [48].

There are two basic problems that prevent T-Cube from being an acceptable form of character input in mobile or wearable devices. First, because of the visual aspect of the pies, eyes-free operation is impossible [33]. Second, circular shaped menus have been known



(A)



(B)

Figure 2.5: (A) Cirrin stroke for the word "soap". (B) Quikwriting stroke for the word "the".

to be difficult to scan with the eye for many users [9], reducing the speed at which they can be correctly accessed.

Two other notable self-disclosing systems that incorporate circular forms are Quikwriting [38] and Cirrin [34]. These two systems are quite similar. Each maps the characters of the alphabet about the perimeter of a circular or rectangular form. Characters are drawn by sliding a stylus from the center of the form to a character (see Figure 2.5). By sliding rather than flicking, users can write entire words with one long stroke, sliding from character to character. Because of the circular nature of these systems, however, they both suffer the same problems as T-Cube.

2.4 MDITIM

In 2000, Isokoski and Raisamo developed the Minimal Device Independent Text Input Method (MDITIM) [23]. MDITIM represented drawings of characters with a chain of the four cardinal directions — North, South, East and West (N, S, E, and W) — (see Figure 2.6(A)). This coarse grain resolution allows for a wide variety of input devices other

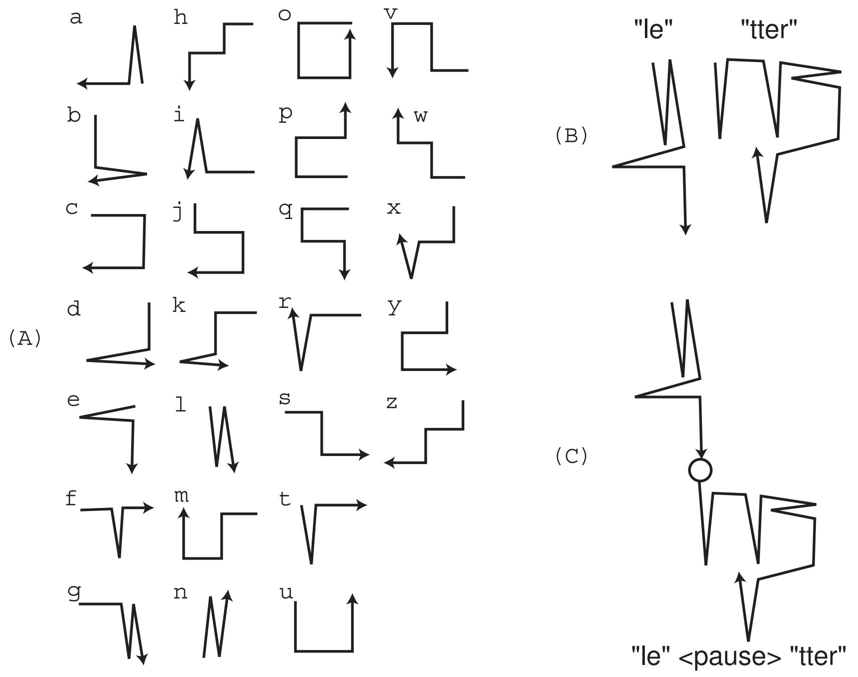


Figure 2.6: (A) Alpha characters of the MDITIM system. (B) The word "letter" drawn with as two separate strokes. (C) The word "letter" drawn as a single stroke with a pause (shown as a circle) to distinguish the consecutive south movements between the first 'e' and 'l'.

than a stylus and pad (e.g., touchpads, mice, joysticks and keyboards). As with Quikwriting and Cirrin, MDITIM allows users to draw entire words with a single, long stroke or with consecutive unistrokes.

No character representations in the MDITIM alphabet include consecutive instances of the same direction. This eliminates any ambiguity that might exist recognizing sequences like ENS and ENSS, where it might be impossible to determine whether the user's intent was one or two 'S's. This is a powerful feature of the alphabet's design; however, this does not eliminate the potential for a multiple character sequence to introduce the same problem. For example, the directional sequence for the word "letter" (SNSWESSNESNEWESWSN) contains an SS pattern on the transition from the first ‘e’ to the ‘t’. Were the SS recognized as a single S, the system would fail at the sequence SNSWESNE and could not recover by any mechanical means. This is because there is no way to determine which of the recognized directions should have been a double. The proper SS would make the SNSWESNE sequence error free, but the second S could also be doubled without introducing errors. . . SNSSWESNE is the valid string “ldt”. To deal with this circumstance, the user may lift or pause the stylus briefly between the consecutive ‘S’s. MDITIM users with trackballs are forced to pause since the ball does have a *lift* analog. When a keypad is used to enter MDITIM strings directions, sequences of key presses are entered rapidly, without pause, because consecutive instances of the same direction are instantly detectable. The use of a keypad with MDITIM additionally makes the system self-disclosing so long as the directional sequences are memorized.

2.5 EdgeWrite

Individuals with nervous or motor impairments are beginning to use mobile devices such as PDAs as controllers or input devices for computers and other equipment [36, 43, 49]. Using a stylus with a PDA has been found to provide a more fluid control experience than a keyboard or mouse for individuals with Muscular Dystrophy, for example [43]. This is because many motor and nervous disorders impair an individual’s ability to make large, rigid movements such as using a mouse [49]. People with Parkinson’s and Cerebral Palsy introduce intention tremors in large movements, and individuals with Muscular Dystrophy lose gross motor control earlier and faster than fine motor control [36].

EdgeWrite is a character recognition technology designed to assist individuals with disabilities that use (or desire to use) PDAs as input devices for computers or other equipment.

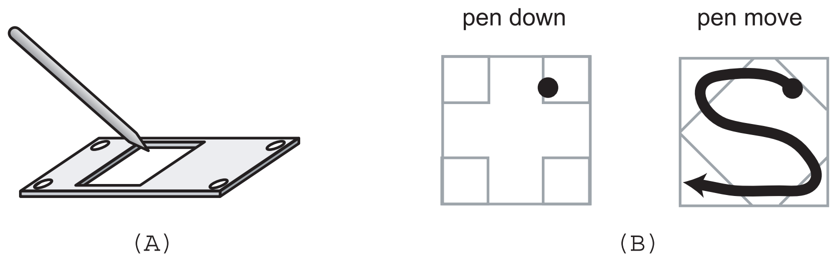


Figure 2.7: (A) The plastic, EdgeWrite template for a Palm PDA. (B) Example corner recognition boundaries over the drawing of the character ‘s’.

EdgeWrite reduces the interference of noise (such as tremor or slipping) by representing characters as a sequence of *corner hits* within a recessed square [49]. Figure 2.7(A) shows a simple, plastic template that be attached to a Palm PDA in order to make it EdgeWrite compatible. Characters are then drawn as a single stroke by touching the stylus to the first corner of the representation and then sliding the stylus from corner to corner over the rest of the drawing. Since the corner sequences are the key to EdgeWrite recognition, impairments that might cause noise over the lengths of the stroke between corner hits will have minimal influence on overall recognition[49]. Many users choose not to fully slide the stylus along the hard edges of the template as the character representations suggest — Figure 2.8 shows the alpha character representations of the EdgeWrite system. Instead they target corner regions (without actually hitting a corner pixel) as part of their stroke resulting in rounded figures that reflect their roman counterparts to a greater extent [49]. Thus, properly determining when and which corners are hit over the length of a stroke is a crucial element to the workings of EdgeWrite. Corners are detected in regions by two separate mechanisms as shown in Figure 2.7(B). When the initial pen down event occurs, corner regions are treated as rectangular zones around each corner. As the stylus begins to move, the corner regions

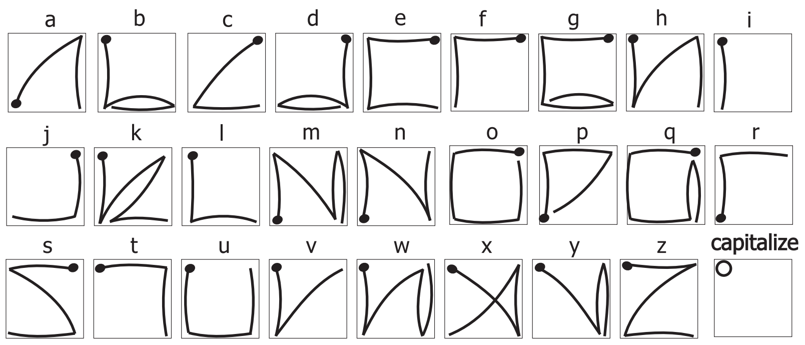


Figure 2.8: The alpha character representations of the EdgeWrite System.

are converted to triangular zones around each corner to reduce the number of unintentional corner hits by users not sliding along the template edges [49].

Wobbrock et al [49] noticed that users targeted corners more liberally on the side of their dominant hand. This is because the stylus is angled toward the dominant hand so the tip can not actually reach the full corner unless the users hand changes its angle for these corners — Figure 2.7(A) shows this issue for a left handed user. Figure 2.7(B) demonstrates how corner recognition zones can be enlarged on the dominant side of a left handed user to further accommodate this problem [49].

2.6 Elastic and Structural Matching

Some of the most robust recognizers in development today are based on elastic and structural matching techniques [3, 5, 11, 12, 20, 31, 47]. While recognition accuracy for these algorithms is somewhat high (averaging 83-98%), their recognition speed can be low.

With elastic matching, drawings are treated as raw sequences of (X,Y) coordinate pairs. Classification of a drawing against an alphabet is done by finding the character instance in the stored alphabet that has the smallest elastic cost when points in the new drawing are *stretched* to match points in the stored instance. This cost between drawings, $\overline{E}$, is the average elastic distance to sequences in the stored instance from those in the new drawing, as in Equation 2.1.

$$\overline{E}(\mu,\lambda) = \frac{E(\mu,\lambda)}{\lambda} \tag{2.1}$$

Here, the new drawing has μ points and the stored instance it's being compared to has λ points. The elastic distance, E, to the new drawing is found and averaged over λ. $E(i,j)$ (defined in Equation 2.2) calculates this distance over the sequence of points in the new instance (starting with point i) and the sequence starting at j in the stored instance.

$$E(i,j) = d(i,j) + \begin{cases} i=0: & \sum_{k=0}^{j-1} d(0,k) \\ j=0: & \sum_{k=0}^{i-1} d(k,0) \\ (i>0),(j=1): & \min\begin{cases} E(i-1,j) \\ E(i-1,j-1) \end{cases} \\ (i>0),(j>1): & \min\begin{cases} E(i-1,j) \\ E(i-1,j-1) \\ E(i-1,j-2) \end{cases} \end{cases} \tag{2.2}$$

$E(i,j)$ is a recursive calculation (hence its tendency to be slow) terminated by the subdistance measure, d, discussed later. On most occasions, the operations of $E(i,j)$ as handled by the last cases in Equation 2.2 are similar to those operators used in traditional string matching techniques. Specifically, extraneous points are identified and removed, missing points are identified and added, and existing points are stretched to match counterparts in the stored instance. The least expensive of these operators is always chosen. The special cases when $i=0$ or $j=0$ indicate that one or the other drawings has run out of points in the recursion. The resulting action is a penalty cost dependant on the k points remaining in the non-emptied sequence. This is where the real *stretching* happens. The subdistance measure, d (Equation 2.3), is the sum of the Euclidian distance and the difference in slope of the drawings tangent to the points in question. The difference in slope is weighted by β, which is chosen by the designer.

$$d(i,j) = (x_i - x_j)^2 + (y_i - y_j)^2 + \beta|s_i - s_j| \tag{2.3}$$

One of the most computationally intense aspects of elastic costing is the fact that $E(i,j)$ must be evaluated many times over the comparison of a single pair of drawings. An optimized approach to this issue is described both by Hellkvist and Tappert [20, 46]. The comparison process is always begun with the $E(0,0)$ calculation which is stored in element $[0,0]$ of a $\mu \times \lambda$ array. By starting i and j with zero values and working upward, the array can be populated such that no calculation is ever repeated during the comparison of two drawings. While storing these values gives an immediate efficiency boost, they must be accessed quite often so the developer must design the code and data flows responsibly to ensure a speedy evaluation of $\overline{E}$ [46].

Merlin [20] was an elastic system developed at Ericsson Radio Systems as the primary means of text entry on their Configurable Phone project. Hellkvist's efforts were primarily on optimizing standard elastic methods for speed as the Configurable Phone's processor was a 133MHz, Intel StrongArm. Merlin specifically focused on a character set including the Graffiti and Jot alphabets. Merlin required just under 150K bytes of runtime and data memories and was recorded at a top speed of 3.03 recognized characters per second.

Experienced Graffiti and Jot users obtained an average accuracy of 97%. Non-experts, however, had a recognition accuracy averaging from 83% and 87%.

Structural approaches to character recognition attempt to extract descriptive, structural strings to represent drawings of characters. This is directly in contrast to elastic techniques which traditionally attack raw coordinate data. Structural representations can include any number of devices, such as directional chain codes (described later in Section 4.1.2), the Printer Description Language (PDL), tree grammars, etc. The activity-based system described in this work extracts structural information in the form of directional chain codes *and* activity measures (see Section 4.2).

Li and Yeung's algorithm [31] incorporates both elastic and structural techniques in a combined recognizer. First a structural analysis takes place, identifying "dominant" points in drawings. A point is considered dominant if it is the elbow point of a 45 degree or greater change in pen direction. The raw point sequence of the drawing is then replaced by the dominant point sequence. This first structural stage works as a pre-classifier and is follwed by the fine classification of elastic matching. The elastic portion of the system works on dominant point sequences rather than raw data. With this system, Li and Yeung reported recognition accuracy averaging 91% and a recognition rate of up to 2.8 characters per second on an Intel 486 50MHz processor.

Chan and Yeung's algorithms [11, 12] incorporate elastic and structural methods in a unique fashion. Drawings are first described in terms of the following structural primitives seen in Figure 2.9: line(dir), up(dir), down(dir), loop, and dot. The up and down primitives represent counter-clockwise and clockwise curves, respectively. A loop is a curve (rotational direction is unimportant) that intersects with itself. The *dir* represents some notion of the direction the primitive ends with... East, Northwest, or South for example. Say there

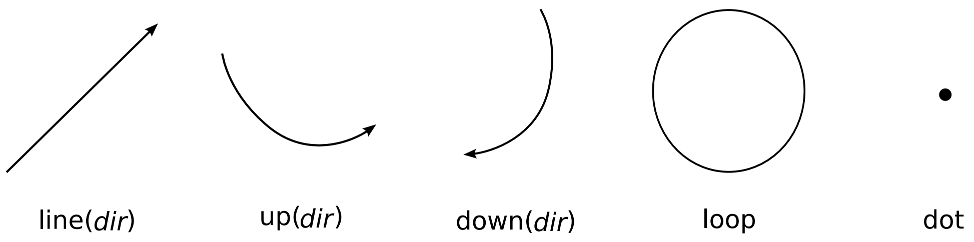


Figure 2.9: Structural primitives employed by Chan and Yeung

are eight directional values considered. In this way there are 8 lines, 8 ups, 8 downs, 1 loop, and 1 dot, totaling 26 possible primitives A drawing is then described as a string of the 26 primitives. Elastic matching is then applied to these sequences where instead of Euclidian distance and slope, $d(i, j)$ from Equation 2.3 can be calculated based on a subdistance matrix between primitives designed by the developer to suit the target alphabet. For example, the distance from line(East) to line(Northeast) may be 2 while the distance from line(East) to up(East) is 1. The developer determines these values to best match the alphabet, preexisting intelligence about its characters, and known deformation tendencies. This provides for extraordinarily high recognition accuracy (98.6% for digits, 98.5% for uppercase, and 97.4% for lowercase [11]), but requires design time intelligence that cannot be updated post-deployment to incorporate new or altered symbols. Recognition speed is, again, moderately slow with an average speed of 7.5 characters per second running on a Sun SPARC 10 Unix workstation. In comparison, the algorithm presented in this paper was timed with an average recognition speed of 16.8 characters per second on the most resource limited implementation — a 20MHz, 8 bit microcontroller without floating-point.

Chapter 3
The Problem of Character Recognition

3.1 e-Studio

The focus of this research originated as part of a 2001, Auburn University project called "e-Studio". The goal of the e-Studio project was to develop a software and network infrastructure to enhance the typical teacher-presentation student-notes experience. Faculty would present slides and handwritten notes over a screen projector, and students would get the same materials delivered to them via any of a variety of networked computers, such as a laptop, PDA, super-phone, tablet computer, etc. These materials would then be accessible at later times for review while on a bus or waiting for the laundry, for example. There was also a desire to promote collaborative environments between the users so that students could present questions, notes, or drawings to faculty from their terminals and create "study groups" to automatically share materials with. This collaborative element is similar to the efforts presented in [29].

e-Studio would provide each user (including faculty members) the ability to add notes to any materials delivered to or received by a terminal — similar to the CrossPad application in the Classroom 2000 project [2]. These notes were generally expected to manifest in two forms — scribbles and text. Scribbles would consist of quick sketches, bullet augmentations, circles, lines, arrows, etc. A typical scribble might be simply drawing a quick star next to an important piece of information or drawing a line to associate physically dislocated bits of information. Text would consist of actual characters and digits that required legibility. There is an emphasis here on legibility because characters and digits could both be represented as scribbles; however, since the resolutions of different terminals may be quite different, a string drawn reasonably on a tablet may appear illegible on a PDA. Thus the text component of e-Studio would provide a means of character recognition so that the content could be stored as strings and rendered appropriately across the various terminal types. My research in character recognition stems from efforts to develop the text element of the e-Studio project.

3.2 Recognition Qualities

A character recognition method designed to satisfy the needs of the e-Studio text element must have numerous qualities. The character recognition algorithm my research presents fulfills each of these.

3.2.1 Low resource usage and portability

e-Studio terminals were expected to include a variety of computing platforms, including inexpensive mobile devices such as super-phones and PDAs. The system would be easiest to expand and maintain if each component (including the character recognition component) were portable across the device gamut. For the character recognition algorithm, this includes the following requirements:

- Memory usage should be minimal, including data stores and runtime memories.
- Recognition must perform in an on-line fashion to ensure an individual's notes are correct. This adds an additional speed requirement to ensure that users are not *waiting* for the recognizer. A recognition speed of 5 characters per second for a Roman-styled

alphabet should suffice considering it would be very difficult for a human to draw characters any faster than this.

- Regardless of a device's input capabilities, any character drawings can be represented or mapped to a two-dimensional picture plane. Thus recognition must be based solely on (X,Y) coordinate data.
- The recognition must support the unistroke drawing standard where each character is drawn to completion one on top of the next. This will guarantee input support for devices such as PDAs and touchpads which are too small to afford characters drawn side by side (as on paper).

3.2.2 Alphabet Independence and User Dependence

The e-Studio system targeted an audience of diverse faculty and students. Users would have different natural and cultural histories, distinguishing the requirement to support multiple, language-alphabets (e.g., English or Cyrillic). However, users of the same nationality may vary in sex, dominant hand, and age (often by generations) and draw characters from the same alphabet in very different ways. Further, users may have developed personal, note-taking shorthand they would like to continue using.

To satisfy these conditions, the recognition system must be tailored to each user (user dependence) and should not inherently respond to characteristics of a specific language-alphabet (alphabet independence). It is important to note that an alphabet independent recognition system need not support every language-alphabet under the sun... not natively at least. Rather, it must be capable of functioning reasonably well given an arbitrary set of drawings as an alphabet. How well is well enough is system and application dependant. If the character recognizer represents the complete system (as on most PDAs), the recognition accuracy must be very high. If instead it is a component of a larger, word-based system (as with those surveyed by [28]), a character-level accuracy of only 70% may be necessary. The user dependant aspect of the system should ensure that character-like markings outside of the user's chosen language-alphabet can be supported in addition to the alphabet's characters... in other words, the user trains the system rather than the user learning the system. This is reasonable since most mobile and wearable devices are typically used solely by the owner.

3.2.3 Revisable Post-Deployment

It is unreasonable (or prohibitively expensive) to expect that a recognition system could be produced to satisfy the issues for alphabet independence and user dependencies for all users prior to deployment. After all, such a system (out of the box) would have to account not only for all language-alphabets, but would additionally support all shorthands and written variants of both. Instead, the e-Studio system should be deployable with some existing character alphabet (optionally) along with the tools necessary to replace, expand, and edit alphabets. This would allow users not only to write in a manner comfortable to them, but it would provide the means to add new shorthand or other characters to the alphabet. Kassel [27] has shown the editing process to be generally acceptable by most users.

To take the alphabet editing, post-deployment, a step further, the system must not require an algorithm update when the alphabet changes, although the particular parameter values for the deployed algorithm may certainly be revised in some automated fashion.

This is a difficult proposition considering recognizers are commonly deployed using some hard-wired bit of human expertise to classify difficult characters [3, 12, 10, 11, 20, 27].

3.2.4 Resistance to Noise

The e-Studio system was to target a wide range of mobile devices. With this in mind, a recognition algorithm suitable for the mobile environment must be capable of dealing with the effects of the environment on the drawing of characters. In particular, regular noise as introduced by say the fairly constant motor of an elevator and isolated or irregular noise (from bumps in the road, for example) should have a minimized influence on recognition accuracy. Performance in a noisy environment should be comparable to that of an otherwise static environment.

3.3 Finding a Sample Corpus for Evaluation

A convenient resource for research scientists in many fields is a common data corpus containing vast amounts of field specific data that can be utilized in experiments and for standardized comparisons of various techniques. For the field field of character recognition, such a corpus would contain samples of several thousand individuals of varying backgrounds, including sloppy samples, along with a digital transcript of the drawings produced by human viewers. While many such repositories exist, none (to my knowledge) are suitable for use in the extended study of on-line, unistroke-style, user dependant recognizers. As such, the experiments presented in Chapter 6 rely on character samples I collected for the purpose of this work.

3.3.1 Off-Line Resources

Of the major character repositories available today, the overwhelming majority are directed specifically at off-line recognition systems. This comes as no surprise since there is such a vast wealth of paper documents that might have an increased value if converted to digital texts… journals, typewriter manuscripts, prescriptions, etc.

One such corpus is the NIST Handprinted Forms and Character Database (Special Database 19) available for purchase over the Internet. It is quite large, containing samples from over 3200 individuals and has been leveraged to develop the recognition systems used by the US Census Bureau. It additionally includes a complete human generated transcript for each sample, as well as database management utilities. Unfortunately there is no temporal information about any of the handwritten documents it contains. Rather, it is based on high resolution (300 dpi) scanned documents.

Since temporal information is virtually always imperative to on-line recognizers (absolutely crucial to the technique described by my work), such databases are useless to on-line researchers. This is a shame since the same wealth of handwritten documents mentioned earlier could be used as the base of new character database for off-line recognizers. In fact, new data sets could be constructed regularly by scanning any of the thousands of handwritten documents that surround us in our everyday lives.

3.3.2 On-Line Resources

There are a few existing resources that are designed specifically for on-line recognition research. To my knowledge, however, non of these are suitable for user-dependant recognizers.

Unipen Database

The first major on-line data corpus was managed by the International Unipen Foundation. This database provides samples from over 2200 writers in the Unipen format. The Unipen format was designed as a standardized means of recording handwritten character samples, far predating similar technologies such as InkML or Microsoft's Journal formats. Samples include information about the writer (eg, name, hand dominance, tablet model) as well as sequenced (X,Y) coordinate data, pen events, and timing information. Like the NIST database, Unipen also includes human generated transcriptions and database tools.

While this database is quite useful and popular, it is not useful for the development of user-dependant recognizers because the subject samples are not controlled adequately. A large percentage of the data does not even include one instance of each character per writer. This is primarily due to the fact that the recording process is not administered and because no standard phrase set is provided to writers. A complete sample for one writer is the word "applesauce". Without adequate frequency of each character per writer it is impossible to separate the data into suitable training and recognition sets. Additionally, the majority of samples are not transcribed by a human reader. A final issue is that there is no control to enforce character segmentation - i.e. a large number of samples include fully connected, script-style characters and ligatures which are not at all suitable for unistroke-style systems where each character is drawn to completion, one on top of the next.

Kassel Data Corpus

For his comparison of recognizers, Kassel devised and collected one of the most substantial databases of handwritten character samples for on-line recognition, which he has since released to the research community [27]. Kassel recorded 159 subjects' handwriting in the Unipen format to ensure compatibility with the existing Unipen tools. Unlike the Unipen database, Kassel developed a standardized phrase set for his experiments containing 599 individual drawings to ensure consistency between subjects. The Kassel phrase set consists of 25 five digit numbers and 54 capitalized words selected from a 20,000 word lexicon, the Merriam-Webster Pocket Dictionary [35]. This provides coverage for upper and lower case English characters as well as digits. Overall, Kassel recorded 95,241 character samples. In particular, Kassel developed his phrase set to be as compact as possible while affording close to English letter-frequencies. The downside to this approach is that the complete sample for any given subject contains too few examples of most characters to be applied to user dependant systems. As seen in Table 3.3, Kassel's phrase set contains only one instance of 13 capital letters, two instances of four capitals, five or less instances of five lower case letters, and 10 or fewer instances of nine lower case letters. There are so few 'G's, once one is used to train the recognizer, only one sample 'G' remains to be tested. This means either 0% or 100% recognition accuracy for 'G'. Further, Kassel intentionally did not control character segmentation, thus fully connected, script-style characters and ligatures exist which are not suitable with Unistroke-style recognizers.

Accountability	Frightfully	Omitted	Taxi
Agonizingly	Fuzz	Projections	Transform
Announcing	Geography	Puff	Uncomfortably
Approaching	Governing	Puzzlement	Unexpected
Backing	Hugging	Quizzically	Unworkable
Cafeteria	Inconsequential	Rejuvenating	Vanquish
Commanding	Industrialized	Revving	Volcanic
Comparatively	Invulnerable	Seeker	Wobble
Complex	Justifications	Shadow	Xylophone
Declaring	Kidding	Skiing	Yearbook
Decompress	Lump	Spoiling	Zero
Disqualified	Mate	Surrounded	
Embraces	Menu	Swab	
Fabulously	Normalization	Sympathetically	

Table 3.1: Words used the the Kassel phrase set

02066	16380	35124	54331
05521	23687	45922	60839
07856	27657	47190	61449
10342	29697	48170	72898
13262	30464	50011	74184
79158	86773	88253	94095
99375			

Table 3.2: Digit sequences used the the Kassel phrase set

‘0’ – 13	‘9’ – 12	‘I’ – 3	‘R’ – 2	‘a’ – 33	‘j’ – 2	‘s’ – 12
‘1’ – 13	‘A’ – 4	‘J’ – 1	‘S’ – 7	‘b’ – 10	‘k’ – 5	‘t’ – 20
‘2’ – 13	‘B’ – 1	‘K’ – 1	‘T’ – 2	‘c’ – 17	‘l’ – 26	‘u’ – 21
‘3’ – 12	‘C’ – 4	‘L’ – 1	‘U’ – 3	‘d’ – 11	‘m’ – 12	‘v’ – 6
‘4’ – 13	‘D’ – 3	‘M’ – 2	‘V’ – 2	‘e’ – 37	‘n’ – 40	‘w’ – 3
‘5’ – 12	‘E’ – 1	‘N’ – 1	‘W’ – 1	‘f’ – 8	‘o’ – 31	‘x’ – 3
‘6’ – 12	‘F’ – 3	‘O’ – 1	‘X’ – 1	‘g’ – 18	‘p’ – 11	‘y’ – 11
‘7’ – 13	‘G’ – 2	‘P’ – 3	‘Y’ – 1	‘h’ – 7	‘q’ – 3	‘z’ – 19
‘8’ – 12	‘H’ – 1	‘Q’ – 1	‘Z’ – 1	‘i’ – 42	‘r’ – 22	

Table 3.3: Character and digit instance counts for the Kassel data corpus

Chapter 4
Activity-Based Recognition

The core of this work is based on a novel feature extraction metric, *activity.* In order for activity to be a useful tool for character recognition, it must be incorporated into a recognizer designed both to feed the metric as well as use its measures to classify handwritten characters. The following sections define the activity metric and introduce a simple recognizer designed to use it.

4.1 Preprocessing

Typically, before any recognition of characters can be performed, a drawing of a character must be preprocessed so that it can be described in the format native to the recognition algorithm. This affords greater recognition accuracy (and perhaps speed) and allows instances of characters to be stored efficiently [20].

4.1.1 Resampling

When drawing a character, it is quite likely that the speed of the pen will vary over different portions of the stroke. For example, while drawing the capital letter 'V', the device capturing the pen movement will probably capture few, well separated coordinates along the left and right slopes, and many tightly packed coordinates around the base joint. This irregular distribution is due to the pen slowing down in anticipation of returning in an upward direction. Additionally, there is no guarantee that the same number of coordinates will be captured each time the same character is drawn.

To deal with these issues, this recognizer resamples the drawing of a character by linearly interpolating $N+1$ Cartesian coordinates into a vector $\vec{R} = \langle r_1, r_2, \ldots, r_{N+1}\rangle$ over the length of the drawing as in [3, 27] so that line segments between consecutive elements in $\vec{R}$ are of equal length (with respect to the traversal length of the original stroke) and both the first and last coordinates are the same as those captured in the original drawing. Figure 4.1 demonstrates this interpolation more clearly. As well as guaranteeing each $\vec{R}$ is of constant size, spatially resampling a drawing in this manner also aids in dampening regular noise and tremor and has been shown to benefit recognition [27]. Figure 4.2 shows four examples of the letter 'G' that are each correctly classified by this recognition algorithm. The leftmost drawing is very close to the character class for 'G' in the test alphabet. The next two examples in the figure were drawn with exaggerated regular noise. Proper classification of these types of drawings is in part due to the noise reduction that resampling provides. Some noise that is introduced into drawings of a character is not regular, say noise that occurs as the result of writing on a bus. Resampling can not be relied on to eliminate this kind of noise. The rightmost drawing of the figure has several instances of this type of noise and is recognizable by the use of the feature extraction method described in Section 4.2, which dampens the noise that spatial resampling can not eliminate.

4.1.2 Directional Codes

While size and position of a drawing on the writing surface could be relevant in enhancing recognition [8], this algorithm emphasizes the direction of pen movement over the course of the stroke. This provides for eyes-free use, where a user is likely to draw the same character in many different locations on the writing surface as well as in varied size. Each consecutive coordinate pair $(r_i, r_{i+1}) \in \vec{R}$ is used to create a vector from the first element of the

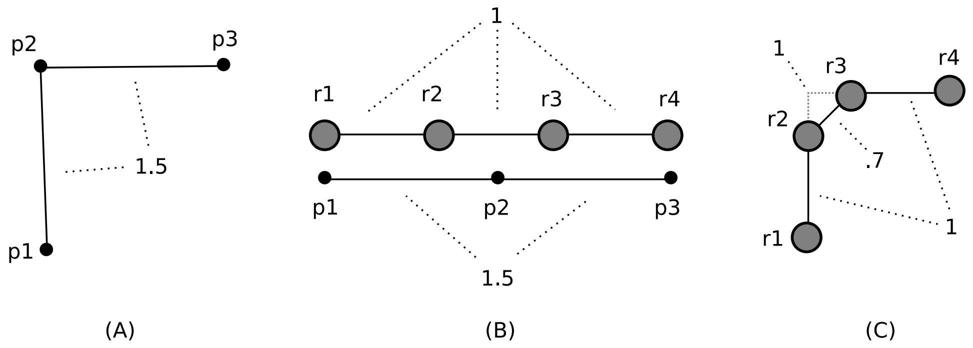


Figure 4.1: Resampling of a simple stroke to four coordinates: (A) Original stroke with three coordinates, (B) Four coordinates placed over the length of the stroke, (C) Final resampled stroke.

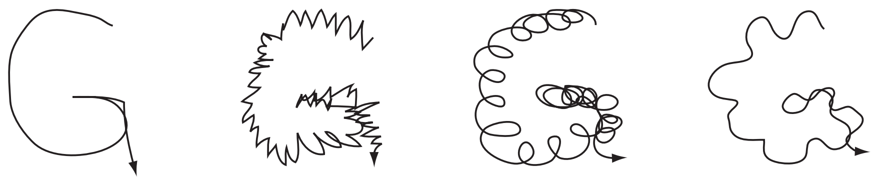


Figure 4.2: Drawings of the letter ‘G’ correctly classified by the presented recognizer

pair to the second. This vector is then mapped to one of a finite number of directional codes stored in a vector $\vec{D} = \langle d_1, d_2, \ldots, d_N \rangle$ where $d_i = \text{DirectionalCodeMapping}(r_i, r_{i+1})$. Freeman's chain code [17] – which divides vector space into the eight cardinal directions E, NE, N, NW, W, SW, S, and SE (enumerated 0,. . . ,7 respectively) as in Figure 4.3(a) – is frequently used for this. Since the presented algorithm was intended to work with custom alphabets, it might also be beneficial to use a generalized direction mapping (based on Freeman's code) so that certain ranges of vector space can be emphasized over others with respect to a particular alphabet and user. Additionally, these ranges can be optimized over an alphabet to further separate characters, thereby improving recognition. For example, if a particular user draws the vertical and horizontal portions of characters in an alphabet in a close to vertical and horizontal manner (with only rare deformations), reducing the ranges for directions 0, 2, 4, and 6 in Freeman's mapping (as in Figure 4.3(b)) may improve recognition accuracy for the user. Further, if few characters in an alphabet require W, SW or S pen movements, the directional mapping could be altered to allow greater discrimination in the other directions, as in Figure 4.3(c). As part of my final experiments, I investigate methods for automating the creation and optimization of directional code mappings on a per user basis. While this can improve recognition accuracy overall when used to prepare directional code vectors, it is beyond the scope of this chapter since it does not alter or accentuate the mechanics of the activity metric.

4.2 Activity

While a vector of Freeman's chain codes could be used alone to describe a drawing of a character, no single vector element can be used to derive information about the overall drawing since deformations tend to be localized. The simple recognizer used throughout

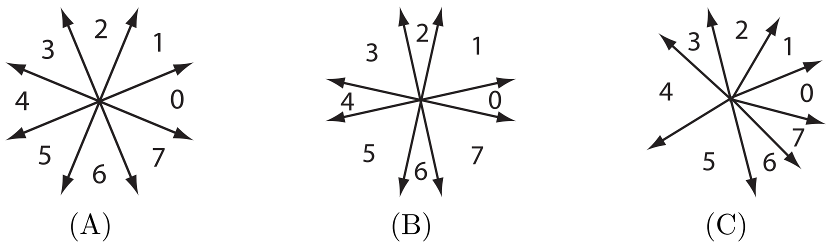


Figure 4.3: Directional code mappings. (A)–Freeman chain code, (B)–Accurate vertical and horizontal lines, (C)–Rarely southwestern

this work attempts to address this issue specifically by introducing a feature extraction metric that further compresses the information gained from directional codes and provides insight into the entire drawing in a general manner. This metric is called *activity* and is defined over a directional code vector $\vec{D}$ as follows:

$$Activity(\vec{D}) = \frac{\text{Length}(\vec{D})}{\text{Dominance}(\vec{D})} \tag{4.1}$$

where Dominance($\vec{D}$) is the cardinality of the dominant (most common) directional code over $\vec{D}$. The activity metric is intended to approximate (quite loosely) the number of unique directional codes required to describe a given vector. If the directional code mapping used enumerates 8 unique values (as in Freeman's chain code), the value of activity over an arbitrary vector of these codes can range generally from 1.0 (only one directional code is present) to 8.0 (all possible codes appear in equal frequency). For example, the directional code vector $\langle 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 7 \rangle$ has an activity of $12/10 = 1.2$. While there are clearly three distinct directional codes in the vector, the non-0 directions are both isolated and could likely be considered noise. The activity measured suggests that the drawing has a single dominant direction with few deformations, thereby significantly dampening noise that remained after spatial resampling. Stating the vector has three different directions, 0, 1 and 7, severely undermines the dominance of 0 and over-emphasizes the presence of 1 and 7.

4.2.1 Activity is When Stuff Happens

In order to understand the reasoning behind the activity metric, you must keep in mind the e-Studio target and desired recognition qualities for which this effort originated (see Section 3.2). An algorithm that seemed particularly promising was developed by Kam-Fai Chan and Dit-Yan Yeung [11] based on elastic structural matching. The primary disadvantages of this algorithm for the e-Studio project was that the runtime complexity of elastic matching was too great for some potential target processors (such as a Zilog Z80). Additionally, similar characters sometimes required the algorithm designer to develop code specifically to distinguish them. For example, the character 'D' could be described as a line in direction N(6) followed by a clockwise curve starting in direction E(0) and ending in direction W(4). Unfortunately, the same description could be used to describe the character 'P'. To resolve conflicts between the two characters, code would be added to calculate the ratio of the height of the curve to the height of the line. Were the ratio below some threshold, the 'P' is recognized, otherwise 'D' is recognized. This eliminates the possibility of modifying an alphabet after deployment.

In an early attempt to correct the inadequacies of the algorithm described in [11], I approached the problem of distinguishing between the characters 'D' and 'P' by drawing each, one after another quickly, with my eyes closed, and then trying to interpret how I knew was drawing one or the other. The key point here was to identify the mentality of *drawing* each character, rather than emphasize how to distinguish drawings of each. The notion I considered was that the difference between the two drawings was "when all the

$$\begin{aligned}
\mathrm{D} \rightarrow\ & \langle 2,2,2,2,2,2,2,2,2,2,2,1,1,0,0,0,7,7,7,7,6,6,6,6,5,5,5,5,4,4,4,4 \rangle \\
\mathrm{P} \rightarrow\ & \langle 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,0,0,7,7,7,6,6,5,5,4,4 \rangle \\
\mathrm{W} \rightarrow\ & \langle 7,7,7,7,7,7,7,7,7,7,1,1,1,1,1,1,7,7,7,7,7,7,1,1,1,1,1,1,1,1,1,1 \rangle \\
\mathrm{V} \rightarrow\ & \langle 7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1 \rangle \\
\mathrm{A} \rightarrow\ & \langle 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7 \rangle
\end{aligned}$$

Figure 4.4: Directional Code representations of 'D', 'P', 'W', 'V' and 'A' of the Graffiti alphabet

interesting stuff happens, or when I deviate from the interesting stuff". To clarify, drawing the 'P' was like drawing a line with a circular tail at the end, whereas drawing the 'D' was more like drawing a semicircle with a line at the head. There is a subtle difference – the first puts the drawing emphasis on the line while the second emphasizes the curve. This examination is what led me to develop the activity metric.

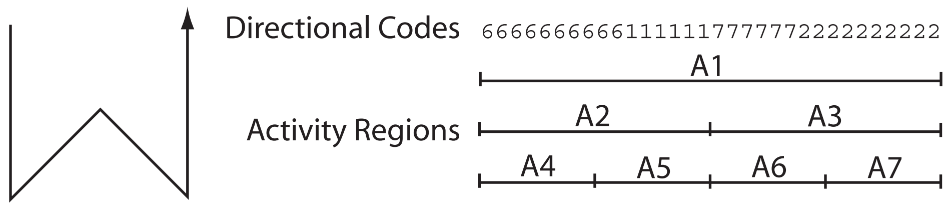
Consider the directional code vectors representing 'D' and 'P' as shown in Figure 4.4. The activity measured over the 'D' is approximately 2.91, while the measure of 'P' is 1.6. These numbers reflect my observations about drawing the two characters. The 'P' is primarily a line in a single direction with deformations (the curve at the end) totaling half the drawing's length, whereas the 'D' is mostly curve. . . a bigger curve mean a higher cardinality of each directional code, and thereby a higher activity. Additionally, notice the curve in 'P' adding 0.6 to the straight line activity (1.0) is consistent with the relationship height of the line and curve ratio measured in [11]. The important thing to recognize here, is that the activity metric does not compare the heights of lines and curves, rather it provides a similar separation measure, for most such problematic character combinations (eg, 'u' and 'y'), but without the need for instance-specific code.

4.2.2 Activity Regions

In order to further increase the usefulness of activity, it is necessary to measure the activity of portions of a drawing rather than only measuring over the entire length of the stroke. Activity regions define these directional subvectors. To this point, only the region spanning the length of the drawing has been considered. In my initial work with the metric I found it beneficial to recognition accuracy to additionally measure activity over regions covering the first and second halves of the drawing, as well as each quarter of the drawing. This totals seven activity regions and is exemplified in Figure 4.5. While the number and location of regions used for a given implementation or alphabet may differ - or perhaps even evolve with usage - I have chosen these seven regions as stock parameters under the assumption that they would be useful for a variety of alphabets. For example, the activity measures over the full drawings for 'W' and 'V' in Figure 4.4 are both 2.0, which does not provide for recognition. Measuring activity on the first halves of each of these characters, 1.6 and 1.0 respectively, and further on the remaining regions clearly separates the two. Additionally, since one activity region may cover a greater portion of the drawing than another or might isolate a particularly revealing portion of the stroke, the activity measured over each region could be biased by some scalar to emphasize the importance of a particular region in distinguishing characters of the current user's alphabet.

4.2.3 When Activity Fails

Regardless of the general success that can be achieved using activity over multiple regions of a drawing, activity fails to aid recognition under certain conditions. Take, for example, measuring the seven activity regions on the characters 'A' and 'V' in Figure 4.4... they are all identical. In fact, *no* region can be defined such that the activity for both characters



A1 = 3.2, A2 = 1.6, A3 = 1.6, A4 = 1.0, A5 = 1.3, A6 = 1.3, A7 = 1.0

Figure 4.5: The seven Activity Regions and measures for ‘W’

is not equivalent. This means that activity alone can not distinguish these two character drawings. The reason for this failure is that activity, while being a measure of direction, in no way reflects direction. A drawing with a full activity of 1.0 has only one directional code present after spatial resampling. What can not be determined from activity is what direction the stroke was in. To resolve this issue, elements of the directional codes must be maintained along with activity so that recognition between these classes of characters is possible.

4.3 Recognition

Deployed use of recognition software based on the presented method takes place in three stages.

1. A character class alphabet is created
2. New drawings are input by the user and converted to the class template used in the alphabet
3. A character in the alphabet is recognized as being the most equivalent (or sufficiently equivalent) to the user's drawings

To prepare a new (custom) alphabet, the user draws each character of the desired alphabet at least once for the recognition system, guaranteeing the character classes in the alphabet contain the irregularities introduced by a given individual's writing style. This affords improved recognition accuracy for the user since the irregularities can be used to further separate characters rather than "test" the classifier in spite of them. Additionally, this method of alphabet generation allows the use of arbitrary, non-Roman characters. The popularity of the Palm OS demonstrates that users are willing to learn an alphabet to improve run time accuracy [42]. It is my belief that users might alternately be responsive to "showing" a device the way they already write.

Each character drawing to be included in the alphabet is defined by an activity vector, a directional code vector and the character associated with the drawing. The inclusion of the directional code vector compensates for activity's lack of directional information (see Section 4.2.3). Care should be should taken when determining the length of each vector to ensure that both direction and activity have appropriate influence in the character classes. For the implementations described in Chapter 5, drawings were preprocessed to a directional vector of length 32. The choice of 32 directional codes was made a priori, and the resulting vector included as the directional code vector in the character. The activity vector used in complement is length 7 over the regions described in Section 4.2.2.

Once an alphabet is constructed, the recognition process is quite straightforward. A new drawing is introduced to the system and described as a directional code and activity vector pair (as above). This character is then compared against each member of the alphabet as a point in multi-dimensional hyperspace (39 dimensional space in my implementations). While I chose Euclidean-squared distance to measure the variance of a drawing and members of the alphabet, other metrics might be equally useful. No studies have been done at this time comparing the quality of recognition gained from alternate distance metrics. Classification over the calculated distances is done with a K nearest-neighbor voting mechanism. The set of K closest character classes is found with respect to a given drawing, and the character with the most (either instance or weight-based) influence over the set is recognized.

Calculating the distance between two characters is simple. The distance between two values on an activity dimension is simply the difference between the two values. The difference between two directional codes is the toral separation between the two codes' values. In other words, the distance between directions 7 and 0 is 1 rather than 7. If a irregular directional code mapping is used, it would be more appropriate to evaluate the distance between codes torally in terms of degrees or radians rather than integer code values. I use this separation approach in my final experiment when I optimize the parameters of recognition for different users.

In an attempt to balance the influence of direction and activity, a scalar bias of 1.222 was applied to each activity measure upon its calculation. This value was determined in the following manner…the range of variance for two Freeman codes is 4, and for two Freeman activities is 7.0, thus the balanced Euclidean-squared influence equation is:

$$7(7.0 \times \text{Bias})^2 = 32(4)^2 \tag{4.2}$$

Chapter 5
Implementations

This work is characterized primarily as the effort to introduce the activity metric and its application to the field of on-line character recognition. The implementations and evaluations of the activity metric resulting from this effort are focused on demonstrating that it is a viable solution for each of the recognition issues described in Section 3.2.

In order to demonstrate the low resource requirements and portability the activity-based recognizer discussed in previous sections, the stock recognizer was implemented and deployed on four platforms: Intel x86, Motorola Dragonball (Palm), Rabbit Semiconductor 2000 (a 20MHz, 8-bit microcontroller with 128K SRAM, 256K flash, and on-board serial I/O), and the Sharp Zaurus handheld. A U.S. patent has been acquired for activity-based character recognition based on these implementations.

The Intel implementation was done first, using Borland C++ Builder on Microsoft Windows. It consisted of an alphabet creation/maintenance application and a notepad-type application for testing recognition. The primary interface of the editor is shown in Figure 5.1. Each character was described as a length 32 vector of directional codes and a length 7 activity vector like that defined in Section 4.2.2. The direction mapping used was the Freeman mapping. A scalar bias of 1.222 was applied to each activity measure upon its calculation.

The small size of the Windows code (only 149 lines of C++, excluding the code for the user interface) and the small data structures required (less than 30K of data) encouraged me to try to implement the algorithm on much smaller, slower processors. Given that

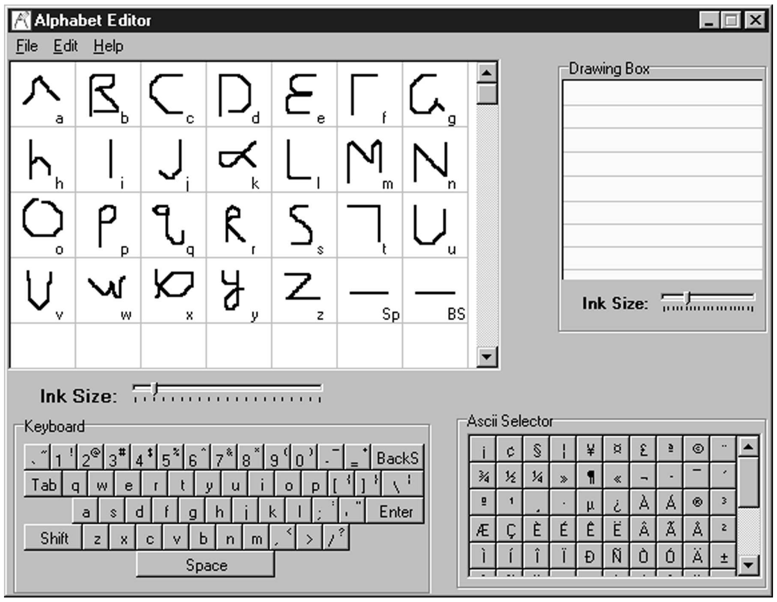


Figure 5.1: Windows alphabet editor

handwriting recognition is now a common feature of PDA's, a fixed-point implementation was developed for Palm OS devices. The parameters used for this algorithm were the same stock parameters used in the Windows implementation other than the modifications required to scale for fixed point.

The Palm implementation requires 35K bytes for code and data, and 6K of persistent storage for an alphabet of 26 characters, space and backspace (all data is unpacked). The recognition screen and alphabet editor screens from the Palm application are depicted in Figure 5.2. While profiling this implementation, it was found the bulk of time spent in recognizing a character was spent during character classification when the distance between members of the alphabet is calculated. As such, this implementation was also optimized for speed by making two intermediate checks of the distance between characters. Since the variance range for an activity measure is twice that of a directional code, the activity vector

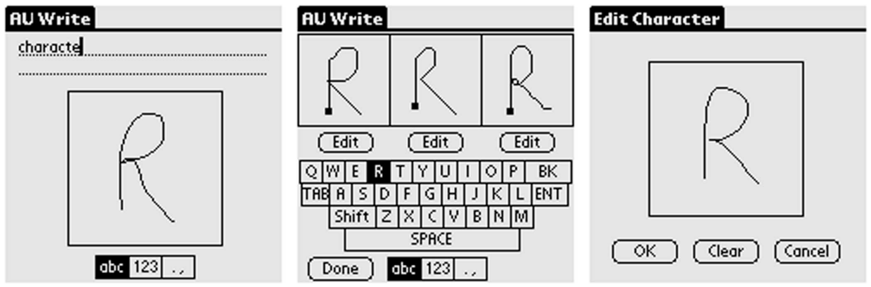


Figure 5.2: Recognition, alphabet, and character editing screens for the Palm OS

is used to form the initial squared sum and a check was made after 12 and 24 dimensions of the direction vector. This allows for terminating the distance calculation if the partially calculated distance is already greater than the distance to the closest character found so far. This resulted in a 22% speed increase at recognition time, based on internal clock measurements.

An 8-bit microcontroller implementation on a 20MHz processor with very small on-board SRAM and flash memories proved the viability of the algorithm for adding character recognition capability to very cheap and very low resource devices. The input device was a Fellowes Touch Mouse and the output device was a 2x20 line LCD display. Code size was 1349 lines of Dynamic C (332 lines for recognition code). Including an alphabet comparable to that used in the Palm OS implementation, the binary image for this application is 40K bytes. No additional memories are required at runtime as no dynamic memory allocation is used. Thus, a processor with a 64K address space is adequate. Measurements using the on-board timer of the Rabbit Semiconductor 2000 indicate a maximum character recognition speed on this very slow device of 16.8 characters per second, significantly faster than humans are capable of drawing characters. This implementation further demonstrates the portability of activity-based recognizers since the Touch Mouse used for input does not

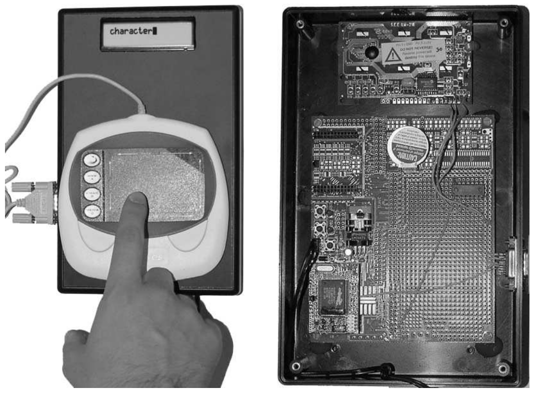


Figure 5.3: Front and back views of the 8-bit microcontroller implementation

report (X,Y) coordinate data, but rather accelerated (X,Y) deltas. This means that a slow motion of one physical inch will report a smaller change in X and Y than a quick motion of one physical inch. The hardware is shown front and back in Figure 5.3. It should be noted that most of the board pictured is an unused prototyping area – the only chips used are the microcontroller, an RS232 driver and an inverter. Due to the limited interface capabilities of this implementation, the alphabet editor written for the Windows environment was used to facilitate the creation of an alphabet. A Perl script was written to convert the files generated by the editor to the binary format required by the Rabbit. These files were then downloaded into flash memory using the Rabbit field utility.

After the first three implementations were completed, a portable, fixed point, ANSI C version of the core recognition tasks (alphabet representation, preprocessing, activity measurement, and character classification) was developed. This core was then used to develop a final implementation for the Sharp Zaurus handheld computer with an interface based on the Palm implementation. The core included 1622 lines of ANSI C code and require 60K bytes of RAM.

Chapter 6

Experiments

6.1 Graffiti Experiments

To measure the accuracy of a simple, activity-based recognizer, a study was performed where 15 university students wrote with the Graffiti alphabet. Graffiti was chosen because many subjects would have some familiarity with it. Additionally, it might be possible to use the results in comparison to the Palm OS, Pocket PC and TealScript (www.tealpoint.com/softscrp.htm) recognizers, each of which support the Graffiti alphabet natively.

Thisphrase set was designed based on the approach used by Kassel [27] as described in Section 3.3.2. A compact phrase set was desirable that reflected English letter frequency but also included many instances of each character to ensure the data was suitable for user in user-dependent systems. Satisfying each of these requirements at once is a mammoth task. First The notion capturing multiple letter-cases or digits with this experiment was eliminated — Graffiti strokes for both upper and lower cases are identical and digits are part of a small (10 characters versus 26) separate alphabet. This reduced the size of Kassel's phrase set by 125 characters, but the real tradeoffs are made trying to balance compactness with letter frequency.

Without regard to case, the letter 'J' appears only three times in Kassel's phrase set — the fewest occurrences of any character. To ensure at least X instances of each character were recorded, Kassel's phrases could be reused minus case but would require subjects to write the set out $X/3$ times. This means to guarantee a minimum of nine instances of each character in the phrase set, subjects would have to draw $(599 - 125) * 3 = 1422$ individual characters (not including any redrawn to account for recognition errors). A few individuals wrote out parts of the phrase set on paper so the time requirements for this size collection could be approximated... about 40 minutes if fatigue did not set in. This is far too long for only nine 'J's but $43 * 3 = 129$ 'I's.

Kassel had dealt with compactness and letter frequency, but because many instances of each character needed to be collected there was little possibility his phrase set could be reused for this study. Further, it was clear letter frequencies would have to loosen to keep the size down and record X instances. My final decision was to construct a phrase set from pangrams, sentences that contain each letter of the alphabet at least once. With this method, choosing the number of pangrams determines X while keeping the phrase set compact. A list of pangrams was compiled from a variety of sources on the Internet and the phrase set was built by selecting the shortest 20. Following are the complete text passages written by subjects:

1. the five boxing wizards jump quickly
2. a very bad quack might jinx zippy fowls
3. a sphinx of black quartz judged my vow
4. a quick brown fox jumps over the lazy dog
5. wavy jake and his fat zebra had mexican pig liquor
6. brawny gods just flocked up to quiz and vex him
7. an exquisite farm wench gave a body jolt to prize stinker

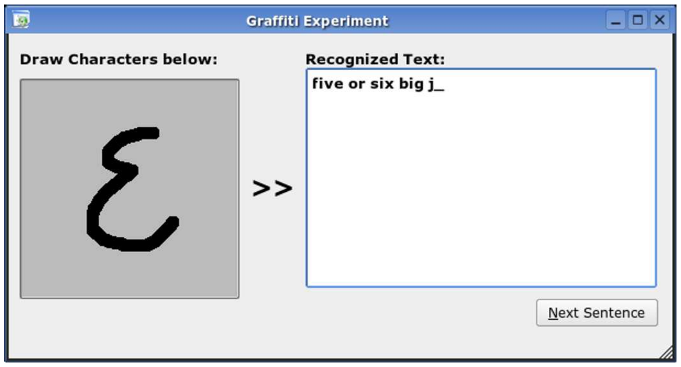
8. five or six big jet planes zoomed quickly by the tower
9. pack my box with five dozen liquor jugs
10. six big devils from japan quickly forgot how to waltz
11. william jex quickly caught five dozen republicans
12. a large fawn jumped quickly over white zinc boxes
13. alfredo just must bring very exciting news to the plaza quickly
14. grumpy wizards make toxic brew for the evil queen and jack
15. back in june we delivered oxygen equipment of the same size
16. jaded zombies acted quaintly but kept driving their oxen forward
17. would you please examine both sizes of the jade figures very quickly
18. six big juicy steaks sizzled in a pan as five workmen left the quarry
19. about sixty codfish eggs will make a quarter pound of very fizzy jelly
20. a mad boxer shot a quick gloved jab to the jaw of his dizzy opponent

This phrase set contains 887 characters (non-space) with instance counts for each character shown in Table 6.1. This is fewer total characters than if the Kassel phrases were doubled while guaranteeing many more instances of each character.

Because this experiment was designed to be subject interactive (subjects see the results of the recognition as they write) and involved English sentences, a simple testing interface was designed similar to a note taking application one might find on a PDA as seen in

'A' – 63	'F' – 25	'K' – 22	'P' – 22	'U' – 40	'Z' – 23
'B' – 23	'G' – 23	'L' – 32	'Q' – 20	'V' – 20	
'C' – 25	'H' – 24	'M' – 39	'R' – 39	'W' – 23	
'D' – 34	'I' – 71	'N' – 37	'S' – 37	'X' – 20	
'E' – 87	'J' – 20	'O' – 55	'T' – 46	'Y' – 29	

Table 6.1: Character instance counts for the Graffiti experiment



five or six big j_

Figure 6.1: The data collection application for the Graffiti experiments

Figure 6.1. The application allowed users to draw characters, one on top of the next in a box on the screen and have the recognized characters appear one after another in a text box to the right.

Subjects were classified as novice (no experience with Graffiti), moderate (having basic comfort with Graffiti) or expert (able to write Graffiti eyes-free). They were each given a sheet of paper with the Graffiti alphabet seen in Figure 2.3 and the complete phrase set for the experiment. Subjects then drew each letter of the alphabet (plus "Backspace" and "Space") three times to train the system using the Windows alphabet editor (Figure 5.1). After this they entered each pangram from the phrase set, pressing the "Next Sentence" button between each pangram.

	Accuracy	# of Subjects
Expert	97.12%	4
Moderate	96.5%	2
Novice	95.01%	9
Overall	95.77%	15

Table 6.2: Average results of the Graffiti study

Subjects were allowed to write at their own pace and were instructed to correct recognized characters by backspacing and redrawing the character. Each backspace was recorded as a character in error. This mechanism allows each subject to determine when a character is misrecognized rather than relying on an automated, character by character comparison of the subjects' text versus the experiment text. As a result, subjects who attempted to memorize phases from the given text but remembered them incorrectly (eg, "the" instead of "this") would not negatively influence the data. Additionally, if a particular subject's drawings of certain characters were difficult to recognize, serial misrecognitions of the same character instance would have an increasingly negative effect on recognition accuracy. This is as close to real world behavior as possible while still maintaining some control over the content.

Recognition accuracy was measured for each subject and averaged across the subject's classification. The results (summarized in Table 6.2) show average recognition accuracies ranging from approximately 95% to 97%. An brief analysis of the data collected from the Graffiti study revealed that the majority of recognition error was the aggregate effect of only several characters being misrecognized frequently. This means the recognizer was generally quite good for all but a few problem characters.

To gain an idea of how activity-based recognition compares to some commercial PDA recognizers, two expert users from the study agreed to repeat the phrase set with the Palm

	Pocket PC	TealScript	Palm OS	Activity-based
Expert 1	94.13%	94.54%	96.2%	98.96%
Expert 2	92.12%	95.03%	95.4%	97.01%

Table 6.3: Accuracy rates of the pilot study with commercial recognizers

OS, Pocket PC (in all-caps mode) and TealScript recognizers. Of the four recognizers evaluated for these users, the activity-based recognizer performed with the greatest accuracy. The results of this pilot study are summarized in Table 6.3.

Several variants of the activity-based recognizer were tried in an attempt to deal with this issue.

One approach that might improve the recognition accuracy of the activity-based algorithm was to divide the recognition comparisons into two phases. First, the activity vector would be used to find some small subset of characters in the alphabet whose activity vectors were the closest to the drawn character. Second, the directional code vector of the drawing would be compared against only those alphabet members found in the activity phase of recognition. This variant is referred to as *activity-first* recognition. Each of these comparisons was done using Euclidean-squared distance.

Figure 6.2 shows how characters in the Graffiti alphabet could begin to be classified based on the number of unique directional codes required to describe the strokes. Only 3 letters are described by a single directional code and 9 are described by two directional codes. Since the activity metric was designed to approximate the number of directional codes that describe a given vector, finding those characters whose activity vector is very similar to that of a given drawing might provide the second phase recognizer with a smaller alphabet of characters with very different directional code vectors. This new, smaller alphabet might then be recognized against using only directional codes, benefiting the overall recognition

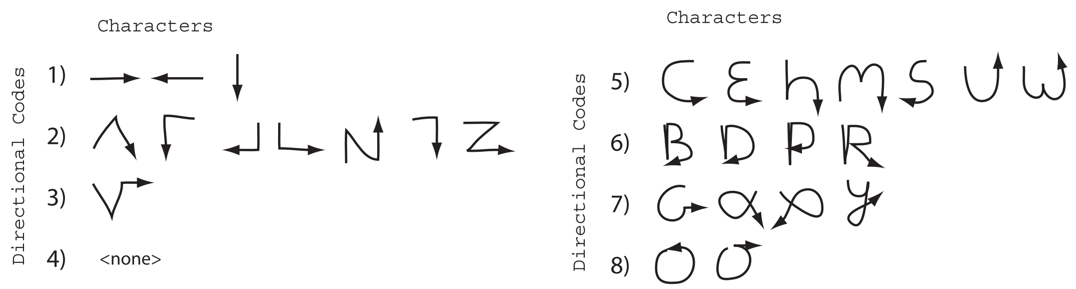


Figure 6.2: Characters of the Graffiti alphabet grouped by total unique directional codes

accuracy, as well as improving recognition speed since only several characters would have the length 32 directional code vectors compared.

Given activity-first recognition, it was thought it might be worth while to reverse the two recognition phases for the sake of comparison. *Direction-first* recognition is implemented by first comparing a drawing with the characters in the alphabet based only on directional-codes. The closest several characters found in this first phase are put into a new alphabet and then recognized against using only activity vectors.

In addition to the previous two variants of activity-based recognizers, two additional recognizers were implemented; *activity-only* and *direction-only* respectively. The first uses activity vectors only to distinguish characters. The second uses only directional codes only – specifically Freeman's chain codes.

To measure the quality of the various recognizers described here, each was used to recognize the 15 subjects' data from the first study. Each recognizer used a directional code vector of length 32 and an activity vector of length 7 spanning the activity regions described in Section 4.2. A scalar bias of 1.222 was applied only to the activity vector of the basic, activity-based recognizer. This is because a bias can not affect the outcome of recognition for the four variants tested. For both the activity-first and direction first recognizers, the

Table 6.4: Average recognition accuracy of five recognizers

	Activity Based	Direction Only	Activity First	Direction First	Activity Only
Expert	97.1%	92.3%	85.6%	77.9%	36.2%
Moderate	96.5%	91.2%	85.7%	74.4%	37.5%
Novice	95.0%	90.2%	83.5%	74.1%	35.8%
Overall	95.8%	90.9%	84.3%	75.1%	36.1%

first phase of recognition generate a new, subset alphabet with 8 members. The results of these experiments are summarized in Table 6.4.

While none of the variant recognizers examined in this paper were able to outperform the basic activity-based recognizer, the results of the experiment are somewhat revealing. First, the direction-only recognizer provided the second best recognition accuracies for this experiment, far exceeding the quality of recognition gained from the activity-only recognizer. This is not surprising since activity is a more coarse grain descriptor than directional codes. Additionally, the activity-first recognizer provided greater recognition accuracy than the direction-first recognizer. This is a reasonable expectation because coarse grain (activity) classification is followed by fine grain (directional) classification. While the activity-first algorithm did not exceed the recognition accuracy of the basic activity-based recognizer, its performance may still be sufficiently improved. Perhaps by applying a unique bias to each activity region in the activity vector, the first phase of the activity-first approach might discover more appropriate sub-alphabets that could improve the recognition accuracy of activity-first recognizers. A similar approach with varying scalar bias could be applied to the activity vector in the basic, activity-based algorithm.

6.2 English Experiment

Having completed the Graffiti study, second experiment was performed focusing on measuring the performance of a stock activity-based recognizer against subjects' non-stylized version of the English alphabet in a non-interactive fashion. This means subjects wrote the text without a recognizer interactively displaying the recognized characters. Instead the same temporal information required by the recognizer (i.e. sequenced (X,Y) coordinate pairs, pen down and up events, etc) was collected so simulate subjects could be used for future optimization studies. Basically, subjects didn't worry about the recognizer's performance so much as they were simply writing text as they might in an eyes-free situation. As with the Graffiti study, the non-stylized study presented users with text passages for them to write, this time with their personal variation of the English alphabet. This affords greater insight into the performance of an activity-based recognizer on character sets other than the Graffiti alphabet which, after all, was designed to be mechanically recognized. Further, the differences between writing styles are much stronger with this study since an alphabet reference sheet could not be provided. Although it would be nice to investigate wildly unique subject alphabets (including the alphabets of languages other than English), it was important to stick with English at this stage so that the content of the captured text could be controlled to a great degree and because many English speaking subjects were available.

A major facet of this study's design was based on the fact a third experiment was planned involving the optimization of the recognizer's parameters. This optimization process would certainly involve many minor and major adjustments to parameter values. After each set of changes were applied, the parameters would have to be evaluated in terms of recognition accuracy. If this study was conducted in the fashion of the Graffiti study, the time and resource expenses involved in having subjects perform the experiment over and over would be unreasonable. Therefore, it was decided to completely reorganize the technique for the sake of the optimization and future studies. The restructuring for the English experiment manifested specifically in two areas: the phrase set, and the fact that subjects would write the phrase set in a non interactive fashion.

6.2.1 Non-interactive collection

The choice to use a non-interactive collection technique ensures that drawings represent the natural style of each subject without recognizer influence. After the Graffiti study was finished and its data reused for introductory tests of variant recognizers, several subjects mentioned that when they encountered consistently misrecognized characters, they altered the way they drew the characters in an attempt to complement the recognizer. This means the results of the variant recognizer tests should be taken with a grain of salt because the data collected was not raw, it was to some degree driven by the original recognizer and therefore not wholly suitable for reuse. It was realized that recording raw, non-interactive subject data would be crucial if one wanted to reuse the data to pursue optimization techniques or investigate alternate algorithms in the future. The non-stylized study would provide such a data store while simultaneously profiling the recognizer with a yet untested alphabet.

Further, many Graffiti subjects indicated the cognitive effort involved in verifying the recognition of each character slowed them down and added some mental fatigue. It was believed, then, that the non-interactive study would go faster and might allow for a greater amount of data to be collected in equal or less time. With this approach recognition would occur at some point after all the drawings had been collected, fed into the system automatically to simulate on-line usage.

6.2.2 Phrase set

The phrase set chosen for the Graffiti study consisted of 20 pangram sentences. This ensured that every letter of the alphabet appeared at least 20 times in the phrase set. The Graffiti alphabet had only one letter case, so in order to reuse these phrases one would have to require that each subject wrote the phrase set twice, once for each case. At first this seems reasonable, but a trial run found it very unnatural to write sentences in upper case. Further, subjects in the Graffiti study often attempted to memorize parts of the pangrams in order to expedite their progress. This resulted in pangrams being transcribed incorrectly — not a big deal when the subject is watching over their own shoulder and can verify the recognition. For a non-interactive mode, however, we had to minimize the possibility that subjects would write the wrong thing because we would have to expect each character they drew was the character requested for the sake of accuracy measurements. Transcription errors introduced by subjects could also waste the effort in designing a phrase set if they result in letter frequencies other than what was intended by the researcher, even if they could be verified by a human reader. A new phrase set was developed that overcame these new issues while adequately satisfying those established in the Graffiti study. It was also a priority to think of a way to address English letter frequencies.

6.2.3 Generating the phrase set

Because the Graffiti phrase set was constructed with little regard to English letter frequencies, we decided to focus on this parameter of the revised collection method first.

After isolating several resources on the topic, it was discovered there are no widely accepted values for letter frequencies or standardized methods for generating new ones. Table 6.5 lists English letter frequencies as reported by three sources, each determined in a unique fashion. The first source (Table 6.5[A]) is the Oxford Dictionary of English [37] which determined its list by counting the letters in words defined in their most recent edition — letters used in definitions, front and back matter were not considered. Lewand [30] (Table 6.5[B]) offers a list suitable for general purpose, English cryptography. Although the sources and collection method he used are unknown, Lewand suggests the most pertinent frequency tables should be constructed by investigators using a representative collection of documents specific to the domain of the material to be evaluated. Linton [32] (Table 6.5[C]) pulls his numbers from three very different contemporary sources: the license agreement from the Sun Java Development Kit 1.2.1, the teaching philosophy of a Computer Science professor from a liberal arts college in Minnesota, and a letter of recommendation for a national competition for innovative uses of technology in collegiate teaching.

Lewand's notion of using domain specific frequencies struck a chord because it is believed the most powerful recognition systems will take application specific information into account to boost performance. Rather than selecting a domain and frequency set, the phrase set was organized so that any frequencies could be soundly applied to the collected data to simulate domain specific frequencies. First, the phrase set must ensure the collection of a statistically large number of each character (30) in upper and lower cases. Additional instances of each letter (both cases) must also be captured for alphabet training. In past efforts we built alphabets with three instances of character. As such we collected three additional instances of each character totaling 33 instances of 26 characters in two cases...

A – 43	F – 9	K – 6	P – 16	U – 19	Z – 1
B – 11	G – 13	L – 28	Q – 1	V – 5	
C – 23	H – 15	M – 15	R – 39	W – 7	
D – 17	I – 38	N – 34	S – 29	X – 1	
E – 57	J – 1	O – 37	T – 35	Y – 9	

(A)

A – 110	F – 30	K – 10	P – 26	U – 37	Z – 1
B – 20	G – 27	L – 54	Q – 1	V – 13	
C – 38	H – 82	M – 33	R – 81	W – 32	
D – 57	I – 94	N – 91	S – 86	X – 2	
E – 172	J – 2	O – 101	T – 122	Y – 27	

(B)

A – 137	F – 39	K – 7	P – 34	U – 48	Z – 1
B – 18	G – 30	L – 75	Q – 2	V – 21	
C – 57	H – 58	M – 47	R – 112	W – 23	
D – 61	I – 128	N – 128	S – 118	X – 4	
E – 207	J – 3	O – 119	T – 162	Y – 32	

(C)

Table 6.5: English letter frequencies (A) reported in *The Oxford Dictionary of English* [37], (B) reported in *Cryptograhical Mathematics* [30], (C) based on three contemporary sources [32]

1,716 samples in all, per subject. Once recognition accuracies are determined for each character, the results can be weighted to match any English frequency set.

Given an arbitrary frequency set $F = \langle f_1, f_2, \ldots, f_{26} \rangle$ where f_i is the relative frequency of letter i (1 being 'A' and 26 being 'Z'), compute the frequency total $F_T = \sum_{i=1}^{26} F_i$. Next calculate the recognition accuracies $R = \langle r_1, r_2, \ldots, r_{26} \rangle$ for each character i. Apply the frequencies to determine the frequency based accuracy A according to Equation 6.1.

$$A = \frac{\sum_{i=1}^{26} f_i \times r_i}{F_T} \tag{6.1}$$

To organize the 858 characters per case into phrases, it would be impossible to use English words, or at least the resulting phrase set would be intolerable. Instead we decided to present the characters in 143, pseudo random strings, six characters long. Then multiple strings would be displayed at one time to subjects, filling out screen after screen of these strings. Organizing these strings completely at random was out of the question, however, because there would likely be character sequences repeating too often to ensure representative variety for each character. It is impossible to ensure no two character sequence is repeated over an ordering of 858 English characters, so it was determined the phrase set for the study would at least contain no duplicate sequence of three characters. To build the final set a primitive algorithm was developed to generate a pseudo random sequence of 858 characters meeting the previously mentioned requirements. First, an 858 length string was randomly populated with 33 instances of each of the 26 characters. Until no three character sequences are duplicated in the string the first character in the repeat sequence was swapped with a random position in the string. Table 6.6 shows the result of this effort, the final phrase set used in the English study.

BASJWD	NJLUMT	URNGTB	MKNUPV	HPROBG	TEVBLC
UKECPM	XPVFKQ	OFMIVA	DQRYVD	XYQAUZ	UMUQXS
VRNIHT	KUGJVS	ZCSJXP	NAJOLQ	VFDILM	GBWCVE
YLGZOX	CYFRZM	KHDYWL	RKYBTP	CTKWHB	YMLKTI
QFGIFK	PTELXD	ZNERCM	MUZFHI	EJNIFD	RFJPDA
RBJVNT	INQWBH	OVXJPY	CGSXEW	GZACTL	HOZNRA
WOMDSQ	AOIATG	AGDSUW	CAQPKD	YUKSQM	ZUCKPF
CZEPUX	MUPHSQ	HFIKQT	JXVUFL	VPWRXO	IHVMDL
YAHLBO	YNOJRW	LBUEIC	RZYGNO	PGHUOI	NGJWTB
WEPJNC	DCLFZB	GTOYLP	BMIWEH	SZFYEB	YXOQES
URGMSH	VXEKGP	BJWNRM	STJKRS	XAMQTN	LEXGVN
DIVKXT	FQWOCI	VFAXZS	TLNIWD	WVKLJC	TAUCJI
YZAFQL	AXNSUK	DKQHWV	HVUCQA	DRUETZ	PZQRDS
IGQKCT	RDZVHB	MSOZHJ	BGFPXM	CJWOGN	BKOMWH
PSUEDH	MLYJET	DEPBCN	ZYOEGM	BAHKDP	YFOBAJ
LVAJFM	BTLEHQ	IXTFRQ	HFBNLT	SRQVFX	SPTKYV
XNRWYZ	WGXSDM	KAYULG	ZVAWSE	YIMLAB	EHDZMU
BOYOQC	NFAIKU	KWMUON	XOKJQI	DWSHPU	LWIGCQ
RINHLA	YPVZCR	CGJEBX	YPRUDC	EOGMNJ	NXFRCZ
FKBGME	OJHBYT	HYLZPT	LBHXPI	QXKYFI	FOJLVS
UPTSDV	CGPWAE	FRSIVD	CVTQAG	ZRLVTC	NDPYIR
ZXJWSC	MUSZND	AQFEWX	EFROYD	NZRDOH	BEHAUG
GZHEBW	RKFOLI	OSGLHC	UMWKJS	AJKPSG	KMTQWX
IORADY	JVQXQE	JBZTIA	ZNSJNE	FYQIWX	

Table 6.6: Generated phrase set for the English character studies

Initially it was a concern that the size of this phrase set was too large to be comfortably performed by subjects in a single sitting. Additionally we were unsure whether the random sequences would result in slower drawing by subjects. To test these concerns a paper test was developed, four pages long and given to five people for profiling. Each page contained four strings from the phrase set. These sheets were laid out identically to the application developed to collect the character samples for this study as seen in Figure 6.3. It was surprising to discover the speed at which subjects filled out these sheets in comparison to the speed of subjects in the Graffiti study. Based on estimates from this profiling it was determined subjects could complete the entire study in close to 50 minutes... only slightly longer than the Graffiti study but with many more character instances. It is speculated the speed increase is based on two things, subjects didn't have to pause between each character to verify recognition, and the pseudo random strings removed any unintentional cognitive overhead occurring if subjects subconsciously interpreted the words in the phrase set.

6.2.4 Collecting Samples

To collect subject samples, two Tablet PCs running the Microsoft XP Tablet Edition were used; one manufactured by Compaq and the other, Toshiba. Both tablets were of the convertible style. A Windows application was developed to collect the character samples as previously outlined. The application (as seen in Figure 6.3) was designed to fill the entire display. Strings from the phrase set were shown in four rows with each character given dedicated screen real estate (a 100x100 pixel box) within which the subject could draw. Each drawing box had a caption (above) that displayed the character in the specific letter-case that drawing should represent. A "Clear" button below each box allowed subjects to erase the data and drawing for a specific character instance if they made a mistake and drew

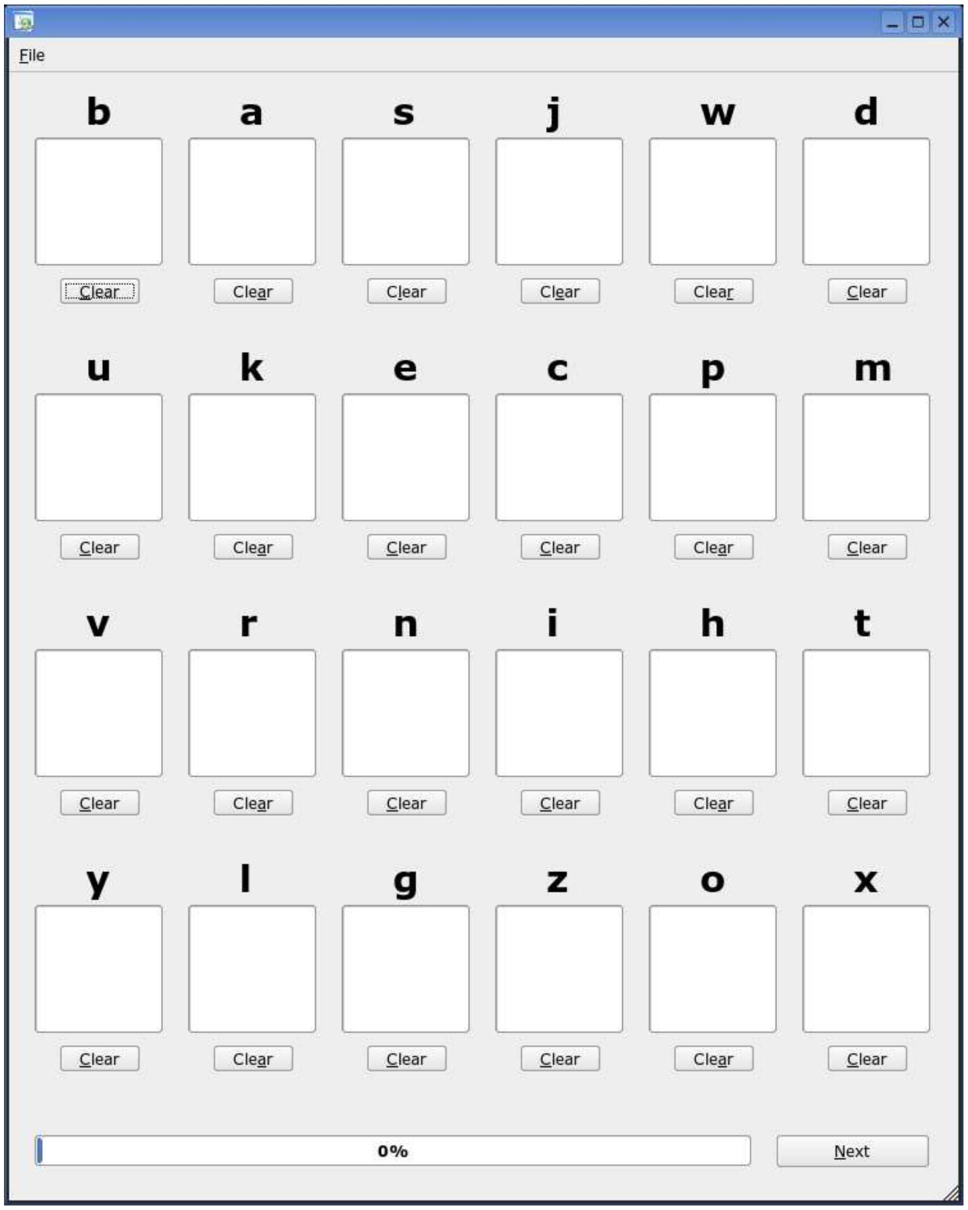


Figure 6.3: Character collection application for the English alphabet studies

the wrong character. The “Next” button on the bottom right of the application replaced the current screen with the next set of strings from the phrase set. The “Next” button was deactivated if any character on the current page had no drawing. A progress bar to the left of the “Next” button showed the subject what percentage of the samples they had completed.

The application presented 72 pages of strings in two halves, each covering the entire phrase set. The first half were lower case. Upon completion of the lower case phrase set, a dialog box appeared letting subjects know the remaining screens would require upper case characters.

66 students and faculty from the Computer Science and Mathematics departments participated in this study. Subjects were given an identifier to ensure their data was anonymous... a letter (‘c’ or ‘t’) representing whether they performed the study on the Compaq or Toshiba tablet followed by a two digit number — eg, “c19”. Although no demographic or subject specifics were recorded with samples, approximately one third of subjects were female. Several subjects wrote with a dominant left hand, and some wrote in Italics (a pseudo-cursive script with disconnected characters). 45 subjects used the Compaq tablet and 21 used the Toshiba.

Subjects were seated in front of a tablet and shown how to use the stylus with the screen. They were allowed to place the tablet in whatever way felt comfortable; eg, resting on the table in front of them or held in their lap. Subjects were given a short demonstration of using the application including drawing in the character boxes, clearing a drawing, and continuing to the next screen. Subjects were instructed to fill the screens based on the way they write with pencil and paper; however, they were additionally asked to maintain a consistent style throughout their samples. Breaks could be taken at any time. On average,

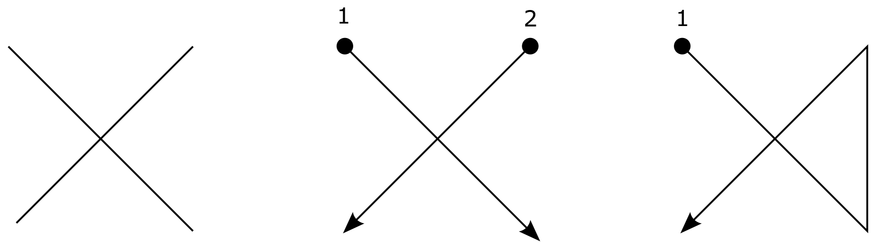


Figure 6.4: From left to right, the letter ‘X’ as drawn on paper, identified as two strokes, and as converted to a single stroke

subjects completed the study in just under one hour. The complete collection of subject samples is located on the packaged CDROM and described in Appendix C.

6.2.5 Alphabet Selection

This experiment is focused on measuring the performance of the stock parameter set identified in Section 6.1 over the upper and lower case English character samples collected in Section 6.2.4. Generally this process is straightforward; however, there were three issues to address before the experiment could continue: handling multiple stroke drawings, determining the size of of the alphabets to be evaluated, and dividing the 33 samples of each character into training (alphabets) and test sets. The issue of multiple strokes can be overcome trivially. It was simply decided to treat all drawings as single strokes by only considering the raw (X,Y) coordinate pairs without regard to whether a visible line would connect them on paper. An example interpretation of the letter ‘X’ is shown in Figure 6.4. The remaining two issues regarding alphabets are not so simple and require further discussion.

In the Graffiti experiment, we had subjects train the recognition system by explicitly drawing three instances of each letter to define the alphabet. For the sake of brevity, we will herein refer to the number of instances of each letter in an alphabet as the alphabet's α value; eg, $\alpha = 2$ indicates the alphabet has two unique drawings of each of the 26 letters of the alphabet, totaling 52 character instances. We had always used an *alpha* value of 3 in our previous work because it provided character variance while resulting in a small enough alphabet to allow fast recognition speed. The samples collected for this experiment allow for a statistically large set of characters (30 per letter) to be tested with $\alpha = 3$. However, because subjects are not explicitly instructing the system to use particular drawings as the alphabet, we could also evaluate the performace of the stock activity system with α values of 1 and 2, testing 32 and 31 samples of the same character, respectively. Thus, for this experiment as well as the following optimization experiment, the system was evaluated using α values from 1 to 3.

Because subjects provided samples as if they were writing on many sheets of paper (rather than into a recognition system) no alphabets were consciously specified. In fact, the collection technique had no concept of of alphabet whatsoever. This raises the question of which characters instances should be chosen for the alphabet and test sets. Kassel [27] (as well as others) chose specific character instances from the phrase set to work as alphabet and test characters. This approach is reasonable considering the small instance counts of many characters, but it only provides a small glimpse into the performance of a recognizer. Because of the large character instance counts provided by subjects, we were able to gain a much more accurate account of the recognizer's accuracy per subject. Instead of a single alphabet and test set, we can generate many random alphabets of a particular α value to determine not only average recognition accuracy, but we can get some idea of the sensitivity of an activity-based system to alphabet selection for a particular subject. This is crucial because this system is intended to be user-dependent.

For each subject we evaluated every permutation of letter-case and alpha value (herein referred to as a *trial*:

Uppercase, $\alpha = 1$	Lowercase, $\alpha = 1$
Uppercase, $\alpha = 2$	Lowercase, $\alpha = 2$
Uppercase, $\alpha = 3$	Lowercase, $\alpha = 3$

Each trial included the random generation and evaluation of 900 α valued alphabets. Initially, the entire sample set for the current case is loaded into a matrix S:

$$S = \begin{bmatrix} s_{1,1} & \cdots & s_{1,33} \\ \vdots & \ddots & \vdots \\ s_{26,1} & \cdots & s_{26,33} \end{bmatrix}$$

where $s_{l,i}$ is the ith instance of the lth letter in the current letter-case. Prior to each of the 900 runs, a random alphabet is generated by swapping elements $s_{l,1}$ through $s_{l,\alpha}$ in S with $s_{l,\mathrm{U}()}$ where U() return a uniformly distributed random integer from 1 to 33, inclusive. This is done for each l (every letter of the English alphabet). The alphabet for this run of the trial is now the first α columns in S, and the remaining columns make up the test set.

6.2.6 Results

Overall, the stock activity-based recognizer performed comparably to its' fellow structural recognition methods (see Section 2.6). With an α value of 3, subjects averaged only a 7.76% error for upper case characters and 8.4% for lower case. The worst recorded errors for $\alpha = 3$ were 27.1% (subject "c11") and 33.78% (subject "c18") for upper and lower case, respectively. The best were 1.48% (subject "c09") and 1.13% (subject "t16") for upper and

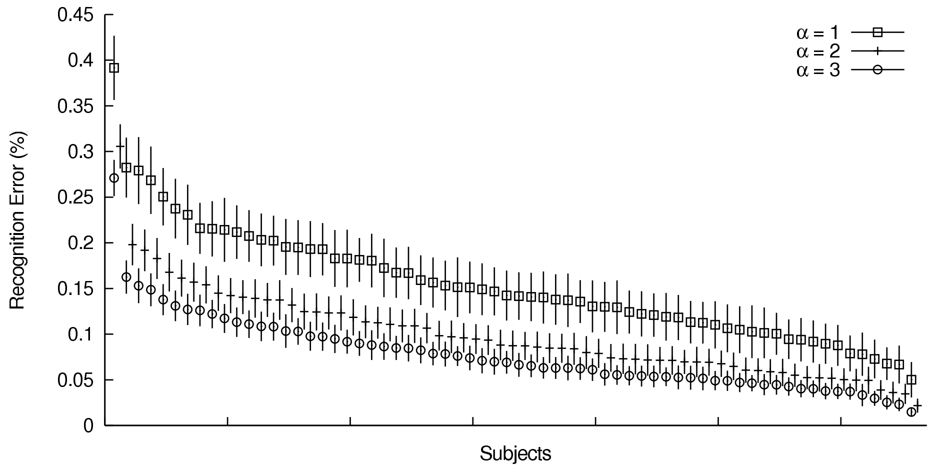
	Upper case		Lower case	
α	mean	σ	mean	σ
1	15.42%	6.11%	16.7%	7.95%
2	9.86%	4.75%	10.68%	6.43%
3	7.76%	4.13%	8.4%	5.65%

Table 6.7: Overall recognition error of the English study with a stock recognizer

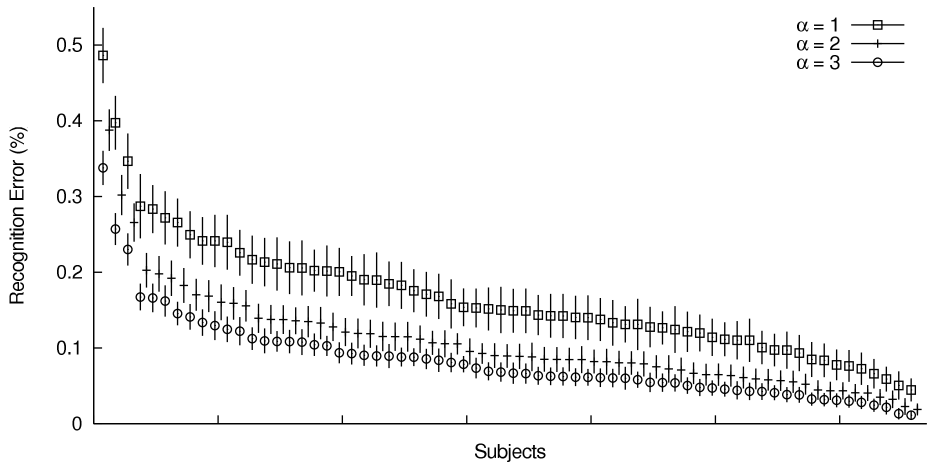
lower case, respectively. Table 6.7 summarizes the average and standard recognition error over all subjects for each α value. The complete results for each subject can be found on the included CDROM described in Appendix C.

Although the error range is quite large, most subjects had good results. Figure 6.5 shows the average recognition errors for each subject trial, sorted. Here it can bee seen that for $\alpha = 3$, 83% of the subjects' error was less than 10% for upper case and 71% of subjects had an error lower than 10% for lower case. What is most striking about this figure, however, is the seemingly minor connection it reveals between average and standard errors. For any given α it is clear the standard error is somewhat larger when the average error is high and somewhat smaller when the average is low. It is also clear that larger α afford smaller standard errors. This is not unexpected. What is unexpected is that for a given α the range of average error may go from 48% to 7% while the standard error differs by around 2%. This is most clearly seen when $\alpha = 1$. This suggests in general that recognition accuracy is not the result of finely tuned alphabet selection nearly so much as it is dependent on the number of instances of each character in the alphabet.

Figure 6.5 shows an asymptotic reduction in error as α increases. The reduction in error was measured over all α transitions for each subject. The results of this evaluation are summarized in Table 6.8. It is clear that increasing the α value of an alphabet will improve recognition, but there is certainly a point at which the accuracy gain is overshadowed by



(A)



(B)

Figure 6.5: Average and standard recognition errors over 900 runs for all subjects in the (A) upper and (B) lower cases.

	Upper case		Lower case	
$(\alpha = i) \rightarrow (\alpha = j)$	mean	σ	mean	σ
$1 \rightarrow 2$	37.79%	5.97%	38.9%	7.08%
$2 \rightarrow 3$	22.67%	4.3%	23.9%	5.7%
$1 \rightarrow 3$	51.67%	7.08%	53.13%	8.66%

Table 6.8: Overall error reduction gained from one α value to another (English study with a stock recognizer)

	Upper case		Lower case	
α	mean	σ	mean	σ
1	15.79%	6.62%	17.13%	7.94%
2	10.0%	5.05%	11.04%	6.49%
3	7.84%	4.35%	8.67%	5.75%

Table 6.9: Overall recognition error of the English study with a stock recognizer and Oxford letter frequencies

the loss in recognition speed. Although it is beyond the scope of this effort, it would be reasonable to develop a formula to predetermine a maximally beneficial *alpha* value for specific implementations that increases recognition accuracy as much as possible while ensuring a minimum recognition speed.

Section 6.2.3 introduces the idea that results from this study could be scaled to an arbitrary letter frequency distribution. To demonstrate this, the Oxford letter frequencies enumerated in Figure 6.5 were applied to the results of each subject in this study. Table 6.9 summarizes the performance of the stock recognizer with the updated letter frequencies.

Figures 6.6 and 6.7 show the recognition error per English letter with respect to α. Thus, the horizontal axes of the figures shows the relative difficulty a stock activity-based recognizer has with each letter. Because the range of average error is so great, the results from two subjects at opposite ends of the accuracy spectrum were visualized in an identical fashion for comparison. Figures 6.8 and 6.9 are taken from subject "c00" whose error was particularly low (but not the best). Figures 6.10 and 6.11 are taken from subject “c02” whose error was particularly poor (but not the worst). Even though the average error of these two subjects is wildly different, it is remarkable to note the similarity in the relative difficulty of characters. For the lower case letters, ‘c’ and ‘o’ are the most successfully recognized letters; ‘g’, ‘i’, and ‘y’ are in the worst five for both subjects. In the upper case characters, both subjects’ best and worst letters (‘S’ and ‘V’ respectively) are identical.

Figure 6.11 also demonstrates an interesting anomaly. Recognition error is quite high on most characters when $\alpha = 1$, but for α values 2 and 3 the error is hyper-reduced. Upon investigation, it was discovered that subject “c02” drew many lower case letters with more than one variation – eg, crossing ‘t’s from left to right sometimes and right to left others. When $\alpha = 1$, no instance of these letters could be selected at random for the run’s alphabet that could ensure recognition across its other instances. When $\alpha > 1$, these letters may now have one of each drawing-style instance in the alphabet. This allows for much broader coverage of the remaining letter instances, and further supports the notion that increasing α may be more lucrative than tuning the alphabet without changing α.

6.3 Optimizing Recognition

To conclude this work, a final study reused the character samples from the English study (Section 6.2) and focused on reducing recognition error for each subject by using a genetic algorithm to optimize the parameters used by the activity metric. Specifically, this optimization study varied directional code mappings, the placement of activity regions, and the scalar bias applied to individual activity regions.

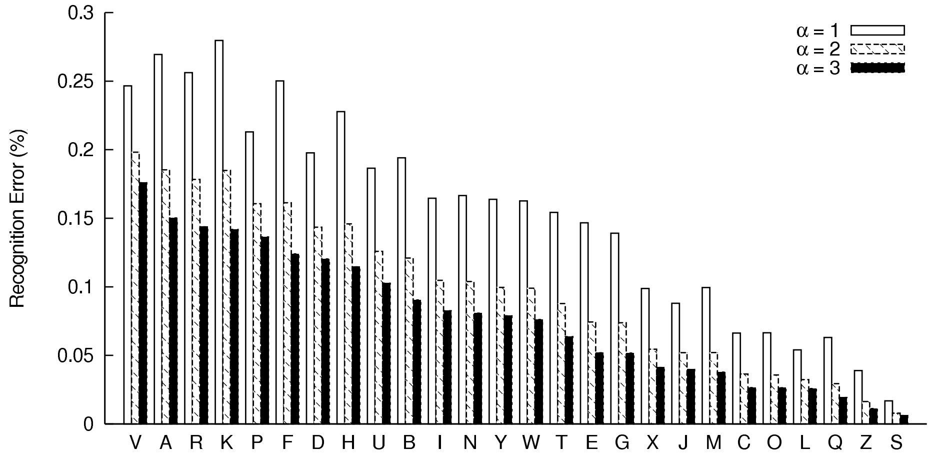


Figure 6.6: Recognition error per uppercase letter for all subjects – sorted by $\alpha = 3$

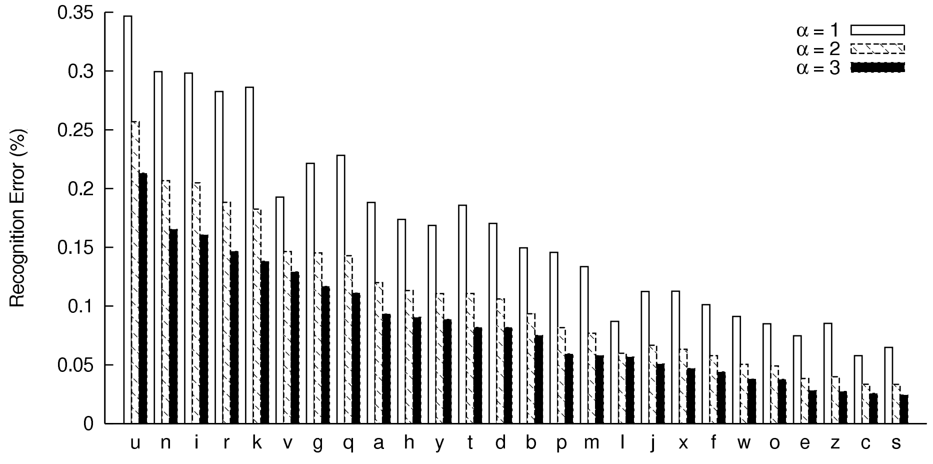


Figure 6.7: Recognition error per lowercase letter for all subjects – sorted by $\alpha = 3$

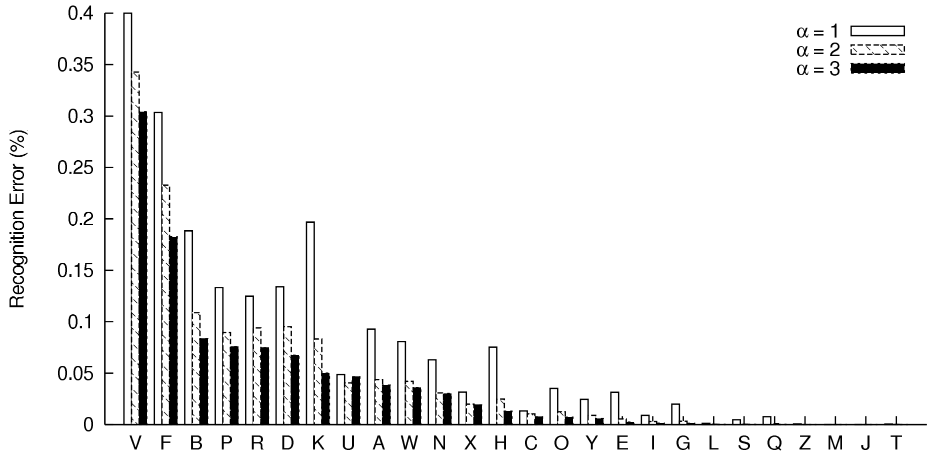


Figure 6.8: Recognition error per uppercase letter for a subject with good general accuracy ("c00") – sorted by $\alpha = 3$

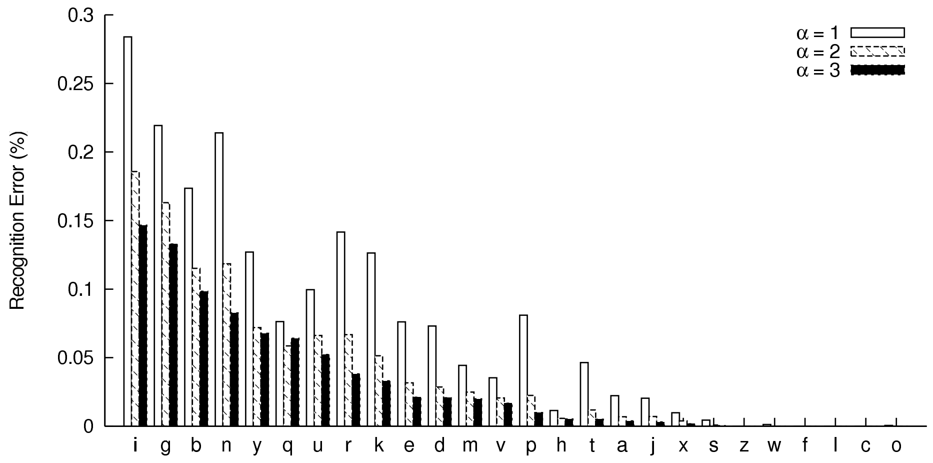


Figure 6.9: Recognition error per lowercase letter for a subject with good general accuracy ("c00") – sorted by $\alpha = 3$

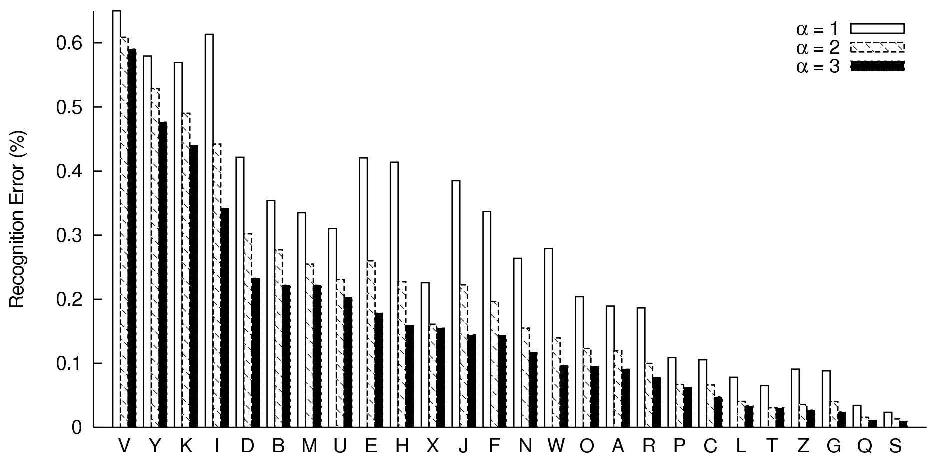


Figure 6.10: Recognition error per uppercase letter for a subject with poor general accuracy ("c02") – sorted by $\alpha = 3$

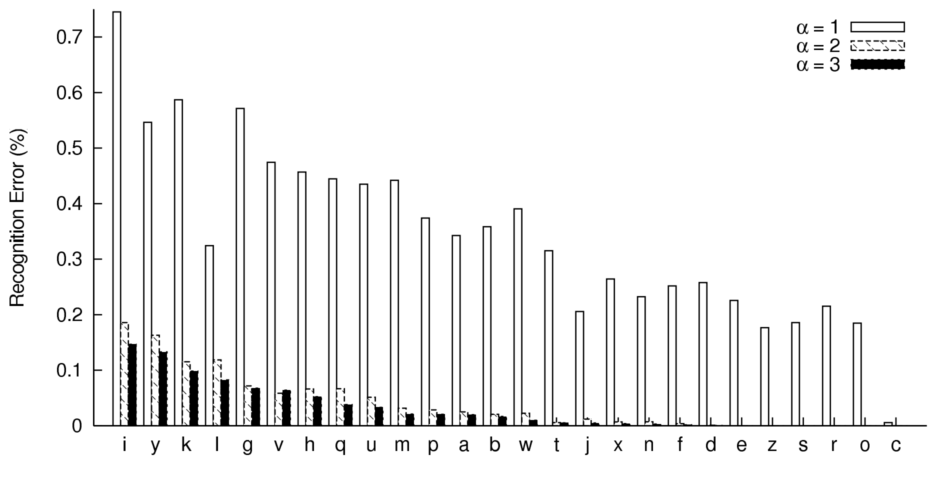


Figure 6.11: Recognition error per lowercase letter for a subject with poor general accuracy ("c02") – sorted by $\alpha = 3$

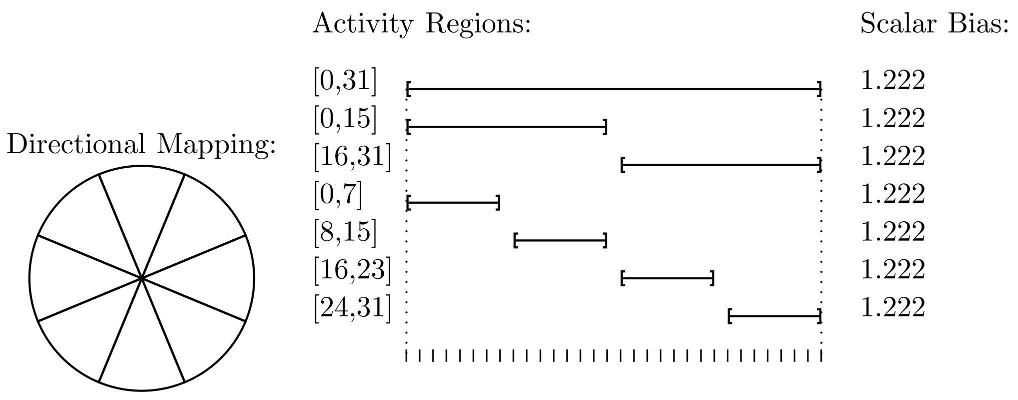


Figure 6.12: Stock parameter set for activity-based systems

6.3.1 Parameters

Figure 6.12 visualizes the complete stock parameter set for activity-based recognition systems as defined throughout Section 4.2. Specifically, a drawing is interpolated into 32 substrokes, each of which is mapped to a Freeman directional code. Seven activity regions are defined and measured over the 32 substrokes as shown below. The 32 directional codes and 7 activity measures form a point in 39 dimensional space. When two character drawings are compared (distance between two points in these 39 dimensions), each of the seven activity measures is scaled by the bias associated with that particular activity region prior to distance calculations. This bias is 1.222 for all regions in the stock parameter set.

More than anything, the purpose of this final study was to demonstrate that elements of the parameters for recognition can be altered to reduce recognition error from the stock set for *every* subject's samples. It was not imperative to actually find the optimum parameter set at this stage... at least not in the traditional sense of optimization. However, for the sake of brevity, these final attempts to reduce recognition error beyond the capabilities of the stock parameters will be herein referred to as optimization. With this in mind, only several recognition parameters were selected to be optimized.

The directional code mapping was the first parameter selected for optimization. While it seems reasonable to use the Freeman mapping, there is no evidence to suggest this mapping maximally separates arbitrary character and symbol drawings. The best thing about Freeman's codes is that a drawing converted to those eight cardinal directions looks very much like the original drawing when replotted using the only the mapped directions. Visual representation, however, does not equate to automated recognition. To optimize the directional code mapping, only the angular regions defining the directional codes were altered, i.e., eight codes were still mapped. No restrictions were set as to where the directional boundaries could lie other than the complete 360 degree circle must be covered such that any angle has one an only one mappable code.

As with the directional mapping, it was decided that seven activity regions should still be measured over 32 interpolated substrokes. The location and size of the regions would instead be optimized. This would allow the regions to lie on those portions of a drawing where the activity metric could provide the greatest benefit. Additionally, the single bias value approach was replaced by a vector of scalars, one unique bias for each of the activity regions. This would allow different regions to affect the separation of drawings with an appropriate level of influence. No restrictions were placed on the location or size of the activity regions. Bias values were restricted only in that they could not be negative.

Selecting this small set of parameters to optimize had the additional benefit of allowing virtually all of the existing recognition code to be reused, unchanged. One thing that did require an update (both in theory and code) was the mechanism used to determine the distance from one directional code to another. Because Freeman's mapping consisted of eight equally sized angular regions, the distance between any two codes had always been the toral integer distance:

$$\text{distance}(i,j) = \min \begin{cases} \max(i,j) - \min(i,j) \\ 8 - \Big(\max(i,j) - \min(i,j)\Big) \end{cases}$$

This mechanism would provide less than desirable results if regions were sized arbitrarily. For example, the distance between any two consecutive directions in Freeman's mapping is 1 using the above distance calculation. With an arbitrary mapping, two consecutive directional centers may be separated by a few degrees while another consecutive pair is separated by 100 degrees. Were the distance between each pair of these codes equal to 1, the recognition system would be ignoring compelling information. To satisfy this new issue, the previous distance measure was augmented as follows where C_x is the angular center (in degrees) of the xth directional region:

$$\text{distance}(i,j) = \min \begin{cases} \max(C_i, C_j) - \min(C_i, C_j) \\ 360.0 - \Big(\max(C_i, C_j) - \min(C_i, C_j)\Big) \end{cases}$$

6.3.2 Genetic Algorithms

Genetic algorithms (GAs) are often used to solve combinatorial and parameter optimization problems and can be implemented into existing systems with ease. They tend to provide close approximations to optimal problem solutions with few resources and in very little time... at least with respect to exhaustive search methods. Because they model the evolutionary process, the solutions they evolve tend to be quite robust. These were the primary motivations behind choosing GAs as the mechanism to optimize the parameters of the activity-based system.

GAs are a form of evolutionary search loosely based on population genetics in the natural world. Holland [21] first introduced the concept as a means to model and study evolutionary processes. Later, Goldberg [18], Eshelman [15], Bäck [6] and others explored the application of GAs to solving optimization problems from various domains.

Essentially, potential problem solutions are generalized to their most basic collection of data structures. This collection is called the *genome*. The elements of a genome are referred to as *alleles* and are encoded in whatever manner is most appropriate to the problem. Holland [21] primarily chose binary string encodings, but Eshelman and Schaffer [15] introduced the use of real-coded (floating point) alleles. Other encodings (such as enumerates) are also possible. Once each of a genome's alleles are given values, that genome instance is referred to as an *individual*. A *population* of individuals is maintained upon which the mechanics of the GA operate. The population undergoes *generations* of evolution in which members of the population known as *parents* breed to create *child* individuals. These children replace population members that die and are removed. The guiding principal here is that if parents are selected to breed based on how well they solve the problem (their *fitness*), they will produce children with above average fitness to enter the population. As this process continues over many generations, the average fitness of individuals in the population will improve, edging the search toward an optimal (or approximate thereof) solution to the problem. If children replace population members based on fitness as well, this process may be further strengthened. Figure 6.13 presents pseudocode for a simple GA.

In Line 1 of the pseudocode, the population of individuals is initialized, and each individual's fitness is measured. This process is most often accomplished by setting each

```
        genetic_algorithm() {
1)          initialize_population();
2)          generation = 0;
3)          until( stop_condition ) {
4)             selection();
5)             crossover();
6)             mutation();
7)             evaluate_children();
8)             replacement();
9)             generation++;
            }
        }
```

Figure 6.13: Pseudocode for a basic genetic algorithm

allele to some appropriate random value. Random bits are used for binary string encodings. There has also been work that suggests *seeding* some alleles with values that are known to be good may improve the speed at which the population improves [26, 41].

Once the initial population is established, the GA begins cycling through generations of the evolutionary process until some predetermined stop condition is reached (Figure 6.13, Line 3). This condition often includes the case of population convergence where each individual in the population is identical. It generally focuses on meeting (or beating) some fitness value or stopping after some predetermined number of generations have past. For problems where fitness is expensive to measure, the total number of fitness evaluations is regularly used rather than generations. This way the population size and parent replacement mechanisms can be altered while keeping the total fitness expense constant.

The first step in a generation is to select the individuals as parents for the next generation (Figure 6.13, Line 4). For each new child desired, two individuals from the population are selected as parents; both parents may actually be the same individual. The total number of children bred at each generation is predetermined by the implementer. The most common means of selection is a probabilistic selection operator where $P(i)$ is the probability individual i (with a fitness of $F(i)$) is selected as a parent:

$$P(i) = \frac{F(i)}{\sum_i F(i)}$$

While this operator is likely the most popular, many other selection operators have been devised [6].

Two parents produce a child by selectively combining allele values in the process known as *crossover* (Figure 6.13, Line 5). The crossover operation is completely dependent on the encoding of individual alleles. When binary coded strings are used, the allele value from only one parent survives in the same allele of the child. From which parent this value is taken must be determined for each allele. If the allele is real-coded, Radcliffe's crossover [40] is commonly used to find a real value somewhere in the space between the value of the allele in each parent. Specifically, if a_1 and a_2 are the allele values of the first and second parents, respectively, child's allele value becomes $a_1 + \beta(a_2 - a_1)$ where β is a real value between 0 and 1, inclusive. β can be calculated uniformly at random or based on the relative fitness of the parents. Figure 6.14 demonstrates each of the operators in a simple example. Eshelman et al [15] extended Radcliffe's crossover to include values just outside the interval between a_1 and a_2. This "blending" crossover, written as BLX-c, computes the child value as $a_1 - c(a_2 - a_1) + \beta\Big(a_2 + c(a_2 - a_1)\Big)$ where c is some real value (see Figure 6.15). BLX-0.0 is equivalent to Radcliffe's crossover operator. Unique operators must be developed for unusually structured alleles. Crossover is often referred to as a global search operator because it can produce wildly unique individuals.

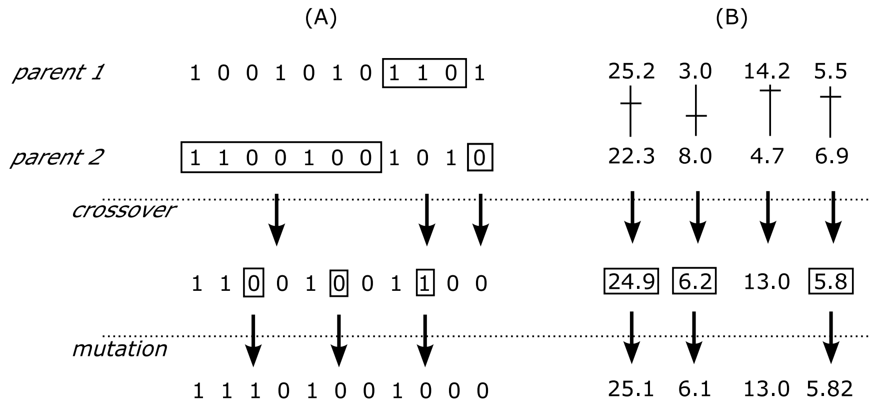


Figure 6.14: Breeding example for (A) binary string and (B) real-coded alleles

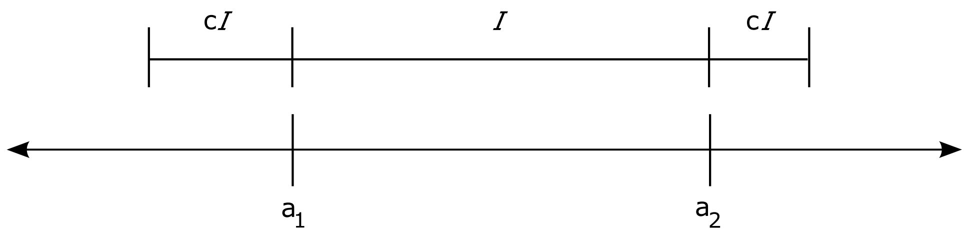


Figure 6.15: The crossover range of the BLX-c operator for real-coded alleles

Once the new child is created, it goes through a process called mutation (Figure 6.13, Line 6) where each allele value has the chance to be "wiggled" a little. This results in a mutated child that is very much like the original child, only marginally different. Mutation is therefore considered a local search operator, particularly once the population begun to converge on a local extrema. For each allele, it must be determined whether it should be mutated. This is frequently accomplished by setting a mutation rate, μ, as the probability each allele should be mutated. Alleles in binary strings are simply replaced by a random bit. Real-coded alleles are offset from their existing value by a normally distributed random number. Figure 6.14 demonstrates each of these mutations. As with crossover, unique mutation operators must be developed for unusually structured alleles.

At this stage each of this generation's finished children has its fitness measured (Figure 6.13, Line 7) and the replacement scheme is activated (Figure 6.13, Line 8). There are two dominant replacement strategies, generational and steady-state. Generational replacement guarantees the entire population dies and is replaced by children at each generation. This method requires that at least as many children are generated as there are individuals in the population. If more children are created the replacement scheme must also determine which of the children are most suitable for the new generation (perhaps using relative fitness). This method allows for great diversity over the generations but is costly in terms of fitness evaluations. With steady-state replacement, only one child is created in a generation and replaces the least fit of the population. This approach is very efficient with respect to fitness evaluations and ensures that individuals with above average fitness remain in the population until they are no longer above average. Bäck [6] introduced an aging metaphor to steady-state replacement to ensure hyperfit individuals cannot remain in the population indefinitely. This mechanism is often crucial in avoiding premature convergence. As individuals enter the population, they are given a time-to-live (TTL) value that is decremented per generation. TTLs are typically large so that the evolutionary process can work naturally. However, if an individual should outlive its TTL, it is immediately removed from the population and replaced with a new individual.

6.3.3 Optimization Operators

In order to optimize the selected parameter set described in Section 6.3.1 using a GA, each of the operators from Section 6.3.1 must be defined. Because so much of the mechanics of GAs are fitness-based, the structural definition and its measurement of fitness should first be disclosed. Each individual represents the parameters for recognizing one letter case for a specific subject.

For the purpose of the GA, a directional code mapping for an individual consists of eight, real-coded alleles, $a_1, \ldots, a_8$, each representing one angular boundary of a single directional code in degrees. When sorted, each consecutive pair (including the last and first elements) completely define the range of a directional code. Each range is specifically measured as inclusive of the most counter-clockwise angle in the pair and non-inclusive of clockwise angle. During population initialization, each of these eight values are set to uniformly distributed random values from 0.0 to 360.0 (non-inclusive) and then sorted. Each of the seven activity regions made up an allele described by two integers, s_i and e_i. s_i is the interpolated substroke on which the ith region begins; e_i ends the region. The activity regions were seeded to initially reflect the stock regions. This choice was made to ensure the regions would have some initial influence suitable to driving the search. Under all circumstances, $s_i <= e_i$. If after some manipulation this does not hold true, their values are immediately swapped to preserve consistency. Finally, eight real-coded scalars, $b_1, \ldots, b_8$, identify the bias applied to the activity measured over the evolved regions. These values were initialized to uniformly distributed random numbers between 0 and 200.

Because recognition accuracy was generally expected to be above 70%, it was decided that the fitness evaluation for this study should register recognition error (generally expected to be low). This provides for a clearer interpretation of relative fitness as accuracy improves. For example, the difference between accuracies of 98% and 99% seems tiny, whereas the equivalent errors (2% and 1% respectively) differ by 100%. Fitness for an individual is measured by evaluating recognition over a subject's specific letter case samples. The evaluation is performed over 300 random $\alpha = 1$ alphabets, similar to the method used to evaluate recognition for a subject in Section 6.2.4. Only 300 alphabets were tested per fitness evaluation to allow for a faster approximation to the actual error. This does mean error rates found in these fitness evaluation are subject to greater variance, but this specific issue is handled by a custom aging operator (described later). Only $\alpha = 1$ alphabets are evaluated under the notion that improving recognition for the known worst α value (as shown in Section 6.2.6) will result in largely improved accuracy for higher α. Since fitness reports recognition error, the GA for this study will attempt to minimize fitness values in its search.

The blending crossover BLX-0.5 was used to crossover angular boundary and bias alleles. A custom operator was required, however, to crossover activity regions. Region alleles would survive into a new child in a fashion similar to crossover in binary strings... the child would inherit each region from a single parent. A crossover probability rate, $\delta = 0.2$, was defined. This rate was used to control the amount off crossover performed in the generation of each new child. Specifically, each allele in the new child was determined as the result of a crossover operation with a probability of δ. Otherwise, the child inherited the allele directly from the first parent.

Angular boundary and bias alleles were mutated using normally distributed random numbers. Angular boundaries were mutated with a standard deviation of 2.5 degrees such that $a_i' = a_i + N(2.5)$. Bias scalars were each mutated with a standard deviation of 5 such that $b_i' = b_i + N(5)$. Another custom operator was developed to mutate activity region alleles. Once it was determined a region allele would be mutated, s_i and e_i were individually updated as $s_i' = s_i + R_1()$ and $e_i' = e_i + R_2()$ where $R()$ returns the values -1, 0, and 1 with equal probability. The result of this operator is that a region either expands, contracts, or slides with varying probability. Figure 6.16 show each of the possible effects resulting from this new operator. To complement the crossover rate, each allele was mutated with a probability of $1 - \delta$.

Parents were chosen from the population using the probabilistic selection operator mentioned above. A modified steady state replacement strategy was chosen for this such that the worst population member was only replaced if the child was more fit. An additional aging element was added to the replacement scheme based on Bäck's work [6]. Bäck was primarily concerned that super individuals could occasionally take control of a population a bring about premature convergence. With this effort, however, there was a greater concern that the sloppy fitness measures (based on 300 rather than 900 alphabets) would allow individuals to remain in the population for extended periods of time based on a fitness value that could actually be misleading. An aging system was needed to ensure individuals would die out of the population once fitness values began to converge. Instead of a deterministic TTL system, aging was managed by increasing the fitness error recorded with each population member by 0.0001 at each generation. What this did was allow individuals to

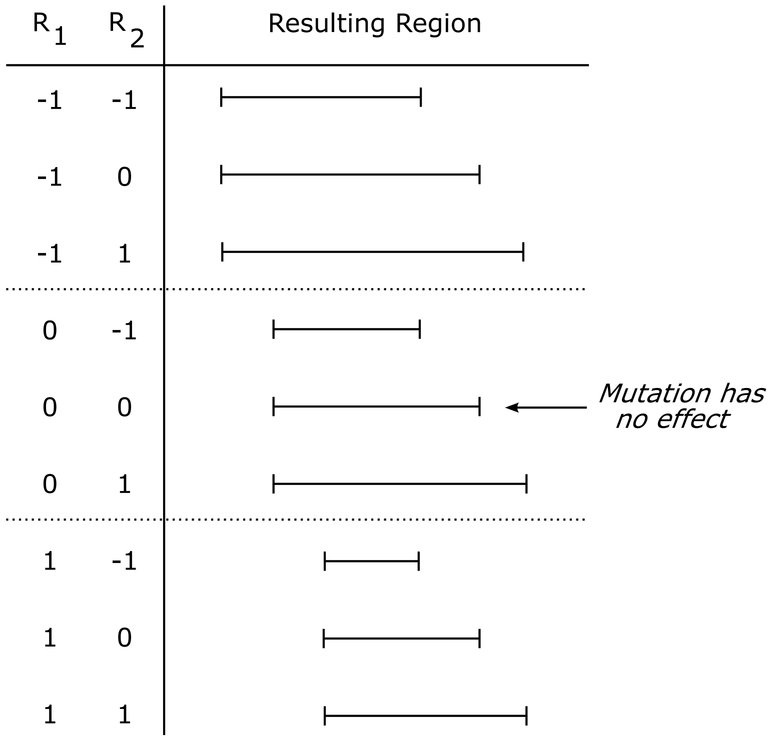


Figure 6.16: Mutation operator for activity regions

remain in the population indefinitely, so long as relative fitness measures were varied within the population. Once the population fitnesses became similar, this aging mechanism would decay the value of any false-positive individuals such that they can be easily replaced by new children.

6.3.4 Genetic Profiling

As the genetic operators for this study were being devised, some concern was raised as to whether activity regions should be evolved at all. The primary issue is there are tight dependency relationships between angular boundaries and activity regions, as well as activity regions and bias scalars. Any evolutionary change in activity regions could potentially interfere with the recognition value of the evolved boundaries and biases. There was also a question as to whether aging might actually be a beneficial operator for this GA. Population size was also undetermined, but some basic numbers were already being considered... 40, 30, 20, 10, and 3 (the smallest usefull population size [18]).

To determine the appropriate answers to these questions, a pilot study was conducted to optimize recognition for subjects "c00" and "c02". Every permutation of population size, aging decay, and whether to use dynamic activity regions was tested. Each of these permutations is listed in Table 6.10 and given a profile letter for reference. Each profile was evaluated on the two subjects for each letter case with a stop condition of 2000 fitness evaluations, chosen arbitrarily.

The performance of each profile was empirically judged on a few factors. In particular, a desirable profile would afford new best solutions throughout the evolutionary process rather than only in tight bursts. Further, it would find high quality solutions (low recognition error) relative to other profiles. Finally, the profile should perform in a consistent manner across

Profile	Population Size	Aging Decay	Activity Regions
A	40	0	Static
B	40	0	Dynamic
C	40	0.0001	Static
D	40	0.0001	Dynamic
E	30	0	Static
F	30	0	Dynamic
G	30	0.0001	Static
H	30	0.0001	Dynamic
I	20	0	Static
J	20	0	Dynamic
K	20	0.0001	Static
L	20	0.0001	Dynamic
M	10	0	Static
N	10	0	Dynamic
O	10	0.0001	Static
P	10	0.0001	Dynamic
Q	3	0	Static
R	3	0	Dynamic
S	3	0.0001	Static
T	3	0.0001	Dynamic

Table 6.10: 20 GA profiles examined for the optimization study

both subjects and letter cases. Upon investigation, Profile T was determined to best meet these requirements and was used to perform the optimization trials over all subjects. The complete results for all the profile runs is included in Appendix A.

Generally, none of the profiles demonstrated any meaningful improvement after approximately 1000 generations. This information was useful in itself as it helped to determine a more appropriate stop condition for the complete optimization trials. As such the final trials were based on the evolutionary results after 1000 fitness evaluations. Additionally, the GA was run on each subject and letter case three times. The best solution from the three is reported as the final, optimized parameter set. The complete list of optimized parameter sets can be found in Appendix B.

6.3.5 Results

Overall, the results from the optimization study were quite good. For $\alpha = 3$, error rate was reduced from the stock results an average of 30.3% and 20.9% for upper and lower cases, respectively. The worst $\alpha = 3$ errors recorded were 24.7% (subject "c11") and 32.07% (subject "c18") for upper and lower cases, respectively. The best were 0.81% (subject "c09") and 0.81% (subject "t16") for upper and lower cases, respectively. Each of these four extreme values are held by the same subjects as reported with stock parameters (see Section 6.2.6). Table 6.11 summarizes the average and standard recognition error over all subjects for each α value. Further, it shows the reduction in average error for each α value per letter case. The complete optimized results for each subject can be found on the included CDROM described in Appendix C.

Figure 6.17 shows the average and standard recognition errors for each subject trial, sorted. Here it can be seen that for $\alpha = 3$, 92% of the subjects' error was less than 10%

	Upper case			Lower case		
α	mean	σ	error reduction	mean	σ	error reduction
1	11.98%	5.94%	24.73%	13.84%	7.84%	20.21%
2	7.3%	4.53%	29.54%	8.7%	6.15%	22.14%
3	5.71%	3.9%	30.32%	6.92%	5.35%	20.92%

Table 6.11: Overall recognition error of the English study with optimized parameter sets

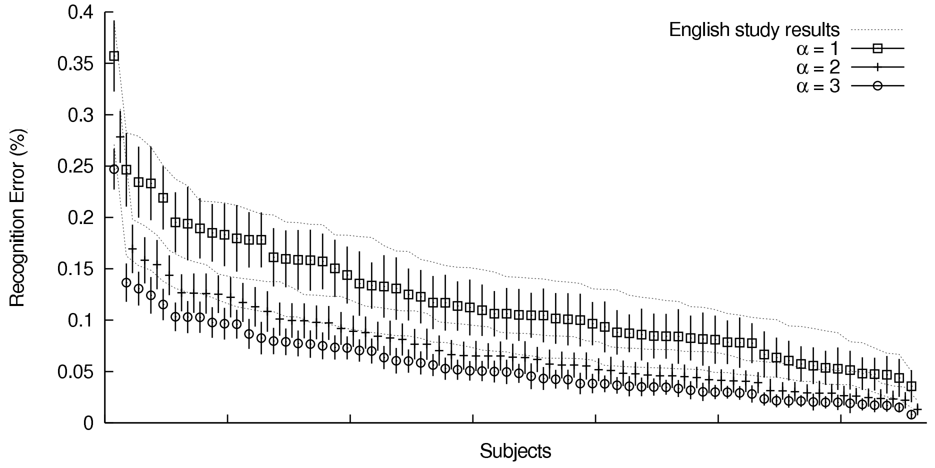
	Upper case		Lower case	
$(\alpha = i) \rightarrow (\alpha = j)$	mean	σ	mean	σ
$1 \rightarrow 2$	42.12%	7.13%	40.59%	7.41%
$2 \rightarrow 3$	23.89%	4.84%	22.84%	4.97%
$1 \rightarrow 3$	55.64%	7.98%	53.84%	8.46%

Table 6.12: Overall error reduction gained from one α value to another (optimization parameter set recognizer)

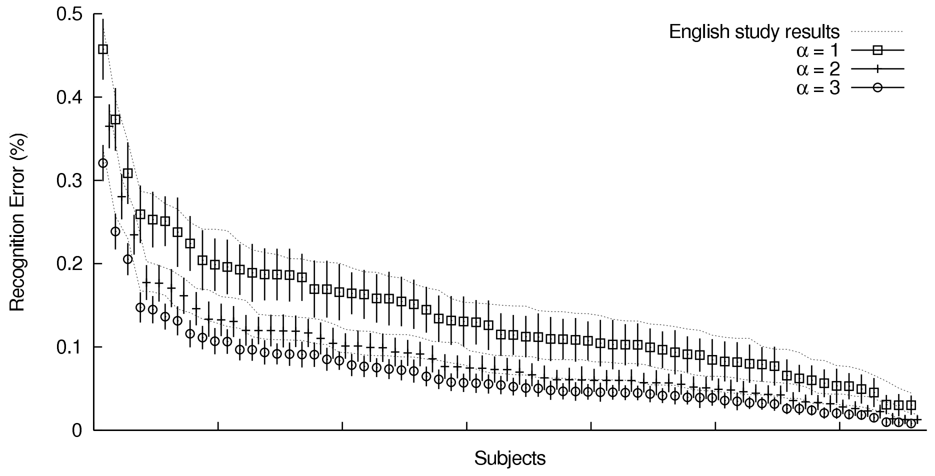
for upper case and 86% of subjects had an error less than 10% for lower case. The thin dotted lines on the figure represent the results for the stock parameter set. It is striking to see for the upper case that the $\alpha = 2$ optimized results were nearly always superior to the $\alpha = 3$ results with stock parameters. Also, for both letter cases and nearly every subject, the stock parameter average is almost always worse than the poor extreme of standard error for the optimized average on the same α.

In Section 6.2.6, it was shown there is an asymptotic reduction in error as α increases, and Table 6.8 summarized this change. For the sake of reference, Table 6.12 treats these same values resulting from the optimization study. It is interesting to note that the reduction in average and standard error is remarkably similar for both the stock and optimized parameters as α increases from 2 to 3.

Section 6.2.3 introduced the idea that results from this study could be scaled to an arbitrary letter frequency distribution. To demonstrate this, the Oxford letter frequencies



(A)



(B)

Figure 6.17: Optimized average and standard recognition errors over 900 runs for all subjects in the (A) upper and (B) lower cases.

	Upper case		Lower case	
α	mean	σ	mean	σ
1	12.65%	6.37%	14.15%	7.94%
2	7.69%	4.88%	8.98%	6.33%
3	6.01%	4.08%	7.15%	5.57%

Table 6.13: Overall recognition error of the optimized parameter set recognizer and Oxford letter frequencies

enumerated in Figure 6.5 were applied to the results of each subject in the stock parameters study. To continue this example, Table 6.13 summarizes the performance of the optimized recognizer with the Oxford letter frequencies.

Figures 6.18 and 6.19 show the optimized recognition error per English letter with respect to α. The horizontal axes of the figures shows the relative difficulty an optimized recognizer has with each letter. Because the range of subjects' average error is so great, the results from two subjects at opposite ends of the accuracy spectrum were visualized in an identical fashion for comparison. Figures 6.20 and 6.21 are taken from subject "c00" whose error was particularly low (but not the best). Figures 6.22 and 6.23 are taken from subject "c02" whose error was particularly poor (but not the worst). Overall, recognition of the letter 'V' improved quite a bit moving it up in the ranking a remarkable 12 positions. For subject "c02", however, recognition of 'V' improved only minimally. On the opposite end of the spectrum, 'P' dropped to the worst upper case letter overall once the parameter sets were optimized. The general ranking of the lower case letters did not change significantly with optimization. For subject "c02", the letter 'r' fell 12 positions in rank.

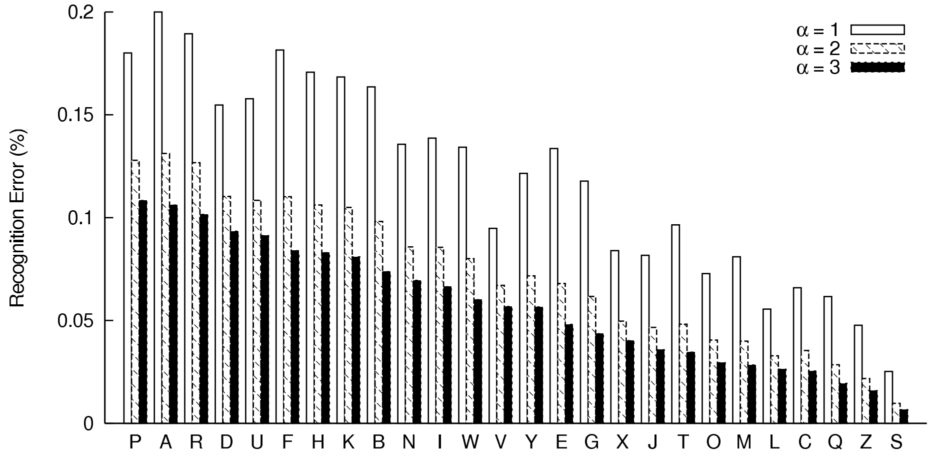


Figure 6.18: Optimized recognition error per uppercase letter for all subjects – sorted by $\alpha = 3$

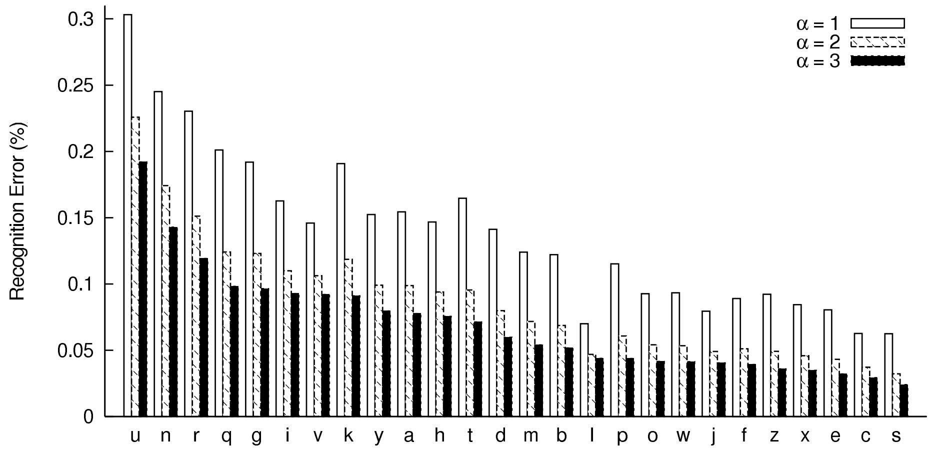


Figure 6.19: Optimized recognition error per lowercase letter for all subjects – sorted by $\alpha = 3$

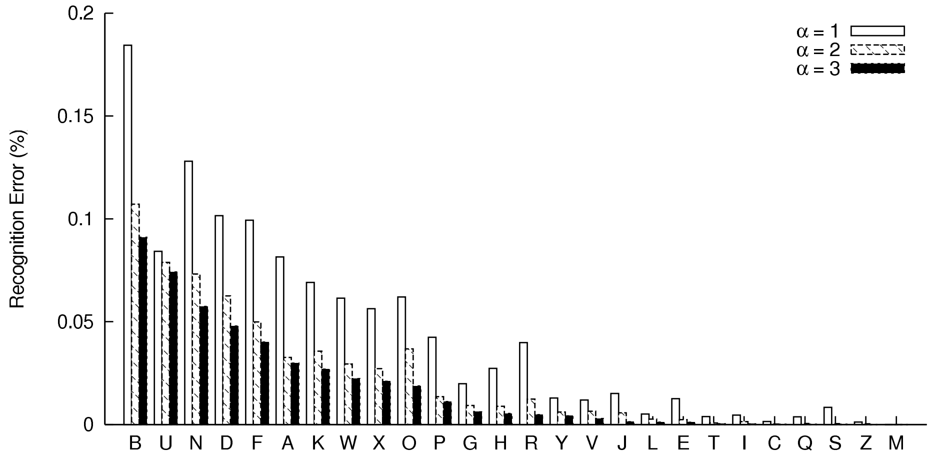


Figure 6.20: Optimized recognition error per uppercase letter for a subject with good general accuracy (“c00”) – sorted by $\alpha = 3$

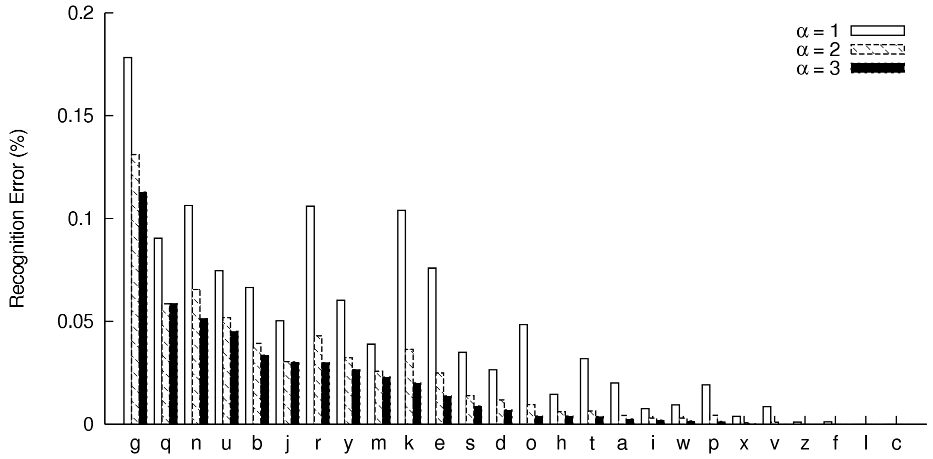


Figure 6.21: Optimized recognition error per lowercase letter for a subject with good general accuracy (“c00”) – sorted by $\alpha = 3$

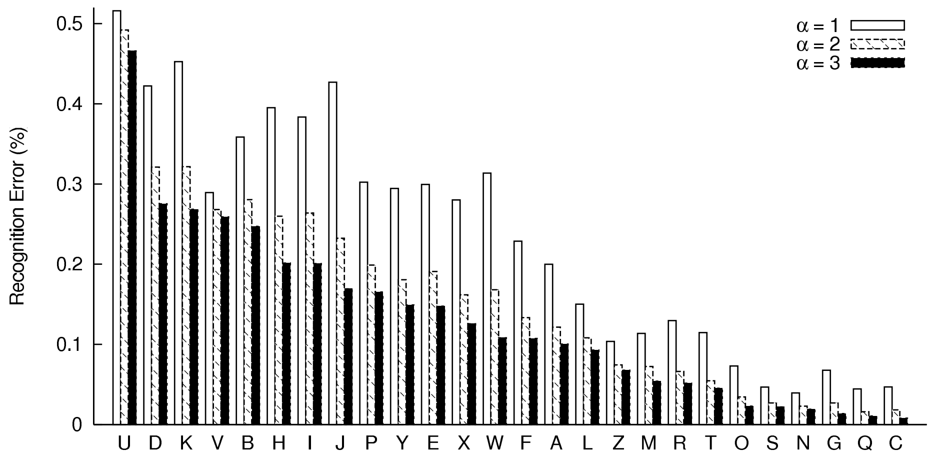


Figure 6.22: Optimized recognition error per uppercase letter for a subject with poor general accuracy ("c02") – sorted by $\alpha = 3$

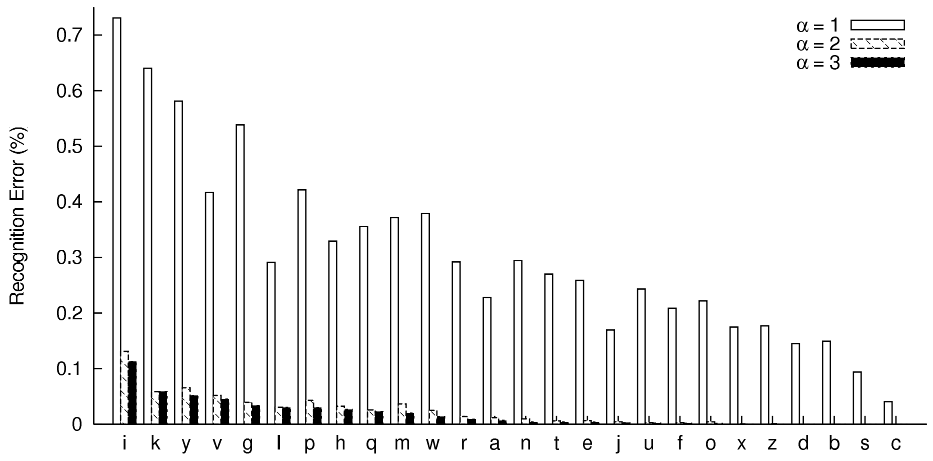


Figure 6.23: Optimized recognition error per lowercase letter for a subject with poor general accuracy ("c02") – sorted by $\alpha = 3$

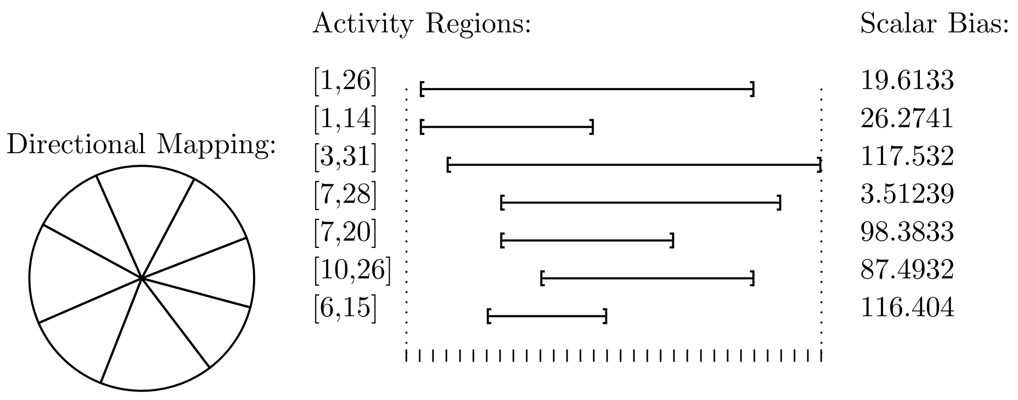


Figure 6.24: Optimized lower case parameters for subject “c21”

6.3.6 Optimization Anomalies

While the results of the optimization study show dramatic recognition improvement over the stock parameters, the most interesting outcome from this study are the evolved parameters themselves.

Subject “c21” reduced lower case, $\alpha = 1$ error by approximately 12% by primarilly altering the activity regions and associated scalars. Figure 6.24 shows the complete optimized parameter set for subject “c21” over the lower case. Notice the directional code mapping is nearly identical to the mapping of the stock parameters. This is by no measure the common case. Most often, all parameters have changed dramatically as a result of optimization.

Figure 6.25 shows optimized directional code mappings for subjects “c10”, “c17”, and “c37”. For each of these subjects, the GA shrunk individual directional ranges to be so small they have essentially been removed from the mapping.
Subjects “c10” and “c17” appear to have only six discernable directions. Subject “c37” has possibly the most astounding example of this range shrinkage with what is basically a five code mapping. Notice also that the mapping for subject “c10” shows clear separation

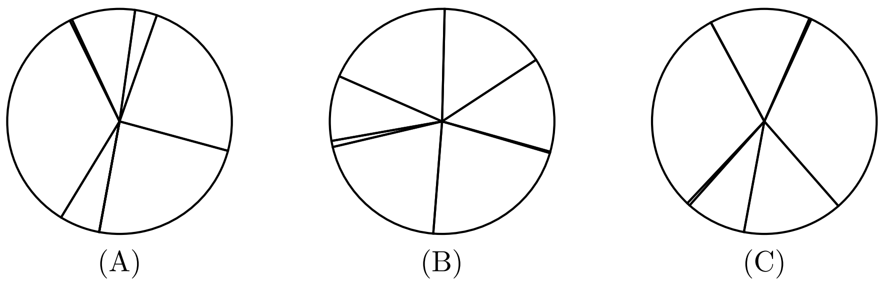


Figure 6.25: Optimized directional code mappings for (A) subject “c10” (lower case), (B) subject “c17” (upper case), and (C) subject “c37” (lower case)

of vertical directions with a typical clockwise slant. On the other hand, the mapping of subject “c17” has no vertical notion whatsoever.

Figure 6.26 shows optimized activity regions for subjects “c29” and “t12”. These two subjects’ regions are exemplary of anomalies seen in many subjects’ optimized regions. Like the previous directional mappings, the GA has evolved the means to remove regions. The upper case regions evolved for subject “c29” have three regions that are nearly identical: each end on substroke 25, two start on substroke 15, and the other starts on 16. Basically, these regions will each measure the essentially the same activity as one and other, regardless of the drawing. Given the three associated biases, b_1, b_2, and b_3, two of these regions could be removed with the remaining region’s bias set to $b_1^2 + b_2^2 + b_3^2$. Although not shown in the figure, one lower case region for subject “t12” has a bias of 0.719. This also has the effect of removing the region alltogether. The evolved upper regions of subject “t12” contain tiny regions covering only two or three substrokes. This may be an “in progress” evolutionary strategy to remove the regions that cut short by the terminating condition of the GA. Were these regions shrunk to where the start and stop substrokes were the same value, the regions would always afford an activity of 1 (Figure 6.27 has three examples of this). However, the

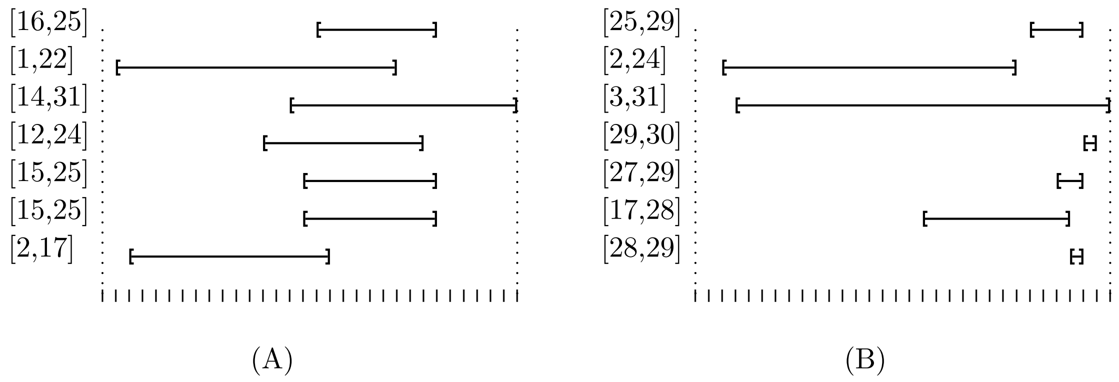
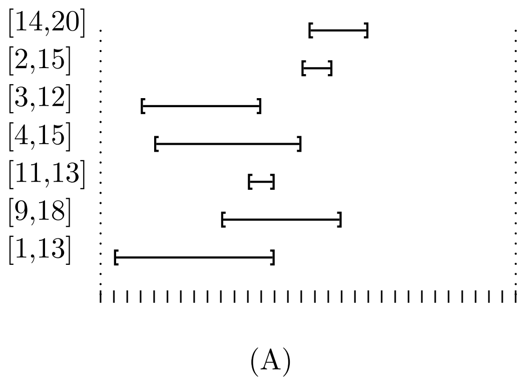


Figure 6.26: Optimized activity regions for (A) subject "c29" (upper case) and (B) subject "t12" (upper case)

non-zero size and relatively close placement of these regions is somewhat peculiar. The two smallest regions can only provide an activity of 1 or 2. Because the bias associated with one of these regions is quite high (122.316), it could be argued that this region plays a crucial, binary roll, identifying whether anything at all is going on in that part of drawing. Even the slightest curve or change in direction would cause the regions to fire high.

For some subjects, entire portions of drawings are shown to have little benefit to recognition. For subject "c05", only the first two thirds of lower case draws appear to have any discerning value. This is seen quite clearly in the evolved activity regions shown in Figure 6.27(A). On the opposite side of the spectrum, the evolved regions for subject "c00" (Figure 6.27(B)) show that only the final third of drawings benefit recognition.



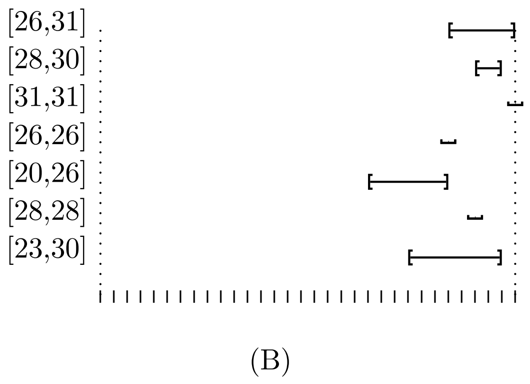


Figure 6.27: Optimized activity regions for (A) subject “c05” (lower case) and (B) subject “c00” (lower case)

Chapter 7
Conclusions

As human-centric interfaces continue to become more and more ubiquitous, there is a greater need to develop methods to provide robust implementations of the most widely used communication mediums: namely, speech and handwritten symbol recognition. This work has described a novel metric, *activity*, to aid in the recognition of handwritten characters. The intent of this metric is not simply to provide another means to do character recognition; rather, it affords the capability to provide high accuracy recognition on even the lowest resource devices. Not only will this allow recognition functionality on devices that have otherwise been without, it can also be leveraged to allow alphabet customization by users even after it has been deployed. Because the metric is based on a few simple parameters (directional code mapping, activity regions and scalar bias) it may be applicable to a wide variety of alphabets and take advantage of user specific idiosyncrasies. The studies conducted and reported in this work provide evidence of this using the Graffiti and English alphabets along with user variants of each. Additionally, a simple, evolutionary method of activity parameter optimization was demonstrated which could be used post-deployment to improve recognition experiences for users. Futher, the interpolated directional mapping has been shown to reduce regular and isolated noise in a fashion beneficial to mobile user who work in shaky or irregular environments, such as a bus, cab, or plane.

The fact that each of these recognition qualities are addressed by such a simple recognition system is what makes this work exciting. As a larger majority of the computer systems we interact with regularly become smaller and more mobile, a recognition system such as the activity-based recognizer detailed in this work will become increasingly valuable.

Bibliography

[1] 3Com. Palmpilot handbook, 1997.

[2] Gregory D. Abowd. Classroom 2000: An experiment with the instrumentation of a living educational environment. *IBM Systems Journal; Special issue on Pervasive Computing*, 38(4), 1999.

[3] Fevzi Alimoğlu. Combining multiple classifiers for pen-based handwritten digit recognition. Master's thesis, Institute of Sciences and Engineering, Boğaziçi University, 1996.

[4] Fevzi Alimoğlu and Ethem Alpaydin. Methods of combining multiple classifiers based on different representations for pen-based handwriting recognition. In *Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN 96)*, June 1996.

[5] Fevzi Alimoğlu and Ethem Alpaydin. Combining multiple classifiers for pen-based handwritten digit recognition. *ELEKTRIK: Turkish Journal of Electrical Engineering and Computer Sciences*, 9(1):1–12, 2001.

[6] Thomas Bäck, Ulrich Hammel, and Hans-Paul Scwefel. Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1), April 1997.

[7] W. Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Proceedings of the EJCC*, pages 225–232, December 1959.

[8] M. Brown and S. Ganapathy. Preprocessing technique for cursive script word recognition. *Pattern Recognition*, 16(5):447–458, 1983.

[9] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman. An empirical comparison of pie vs. linear menus. In *Conference proceedings on Human factors in computing systems*, pages 95–100, May 1988.

[10] Kam-Fai Chan and Dit-Yan Yeung. Elastic structural matching for on-line handwritten alphanumeric character recognition. In *Proceedings of the Fourteenth International Conference on Pattern Recognition*, pages 1508–1511, August 1998.

[11] Kam-Fai Chan and Dit-Yan Yeung. A simple yet robust structural approach for recognizing on-line handwritten alphanumerical characters. In *Proceedings of the Sixth International Workshop on Frontiers in Handwriting Recognition*, pages 229–238, August 1998.

[12] Kam-Fai Chan and Dit-Yan Yeung. Recognizing on-line handwritten alphanumeric characters through flexible structural matching. *Pattern Recognition*, 32(1):1099–1114, July 1999.

[13] C. K. Chow. Optimal character recognition system using decision functions. In *IRE Transactions on Electronic Computers*, volume 6, pages 247–254, August 1957.

[14] J. T. Chu. Optimal decision functions for computer character recognition. *Journal of the ACM*, 12(2):213–226, April 1965.

[15] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann, 1993.

[16] I. Flores. An optimum character recognition system using decision functions. *IRE Transactions on Electronic Computers*, 7(2), June 1958.

[17] Herbert Freeman. Computer processing of line-drawing images. *ACM Computing Surveys*, 6(1):57–97, March 1974.

[18] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[19] David Goldberg and Cate Richardson. Touch-typing with a stylus. In *Proceedings of the INTERCHI'93 Conference on Human Factors in Computing Systems*, pages 80–87. ACM, April 1993.

[20] Stefan Hellkvist. On-line character recognition on small hand-held terminals using elastic structural matching. Master's thesis, Royal Institute of Technology, Stockholm, Department of Numerical Analysis and Computing Science, 1999.

[21] J. H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

[22] Poika Isokoski. Model for unistroke writing time. In *Proceedings of the SIG-CHI on Human factors in computing systems*, pages 357–364. ACM, March 2001.

[23] Poika Isokoski and Roope Raisamo. Device independent text input: A rationale and an example. In *Proceedings of the Working Conference on Advanced Visual Interfaces AVI2000*, pages 76–83. ACM, 2000.

[24] Allan Long Jr., James Landay, and Lawrence Rowe. Pda and gesture use in practice: insights for designers of pen-absed user interfaces. Technical Report UCB//CSD-97-976, U.C. Berkley, 1997.

[25] Allan Long Jr., James Landay, Lawrence Rowe, and Joseph Michiels. Visual similarity of pen gestures. In *Proceedings of Human Factors in Computer Systems (SIGCHI)*, April 2000.

[26] A. Kapsalisand, V. J. Rayward-Smith, and G. D. Smith. Solving the graphical steiner tree problem using genetic algorithms. *Journal of the Operational Research Society*, 44(4):397–406, April 1993.

[27] Howard Kassel. A comparison of approaches to on-line handwritten character recognition. Master's thesis, Massachusetts Institute of Technology, June 1995.

[28] A. L. Koerich, R. Sabourin, and C. Y. Suen. Large vocabulary off-line handwriting recognition: A survey. *Pattern Analysis Application*, 6:97–121, 2003.

[29] James Landay. Using note-taking appliances for student to student collaboration. In *Frontiers in Education Conference, FIE '99*, volume 2, pages 12C4/15–12C4/20, November 1999.

[30] Robert Edward Lewand. *Cryptographical Mathematics*. Mathematical Association of America Press, 2000.

[31] Xiaolin Li and Dit-Yan Yeung. On-line handwritten alphanumeric character recognition using feature sequences. In *Proceedings of the ICSC*, pages 197–204, 1997.

[32] Tom Linton. English letter frequencies. `http://www.central.edu/homepages/LintonT/classes/spring01/cryptography/letterfreq.html`, 2001.

[33] Scott MacKenzie and Larry Chang. A performance comparison of two handwriting recognizers. *Interacting with Computers*, 11:283–297, 1999.

[34] Jennifer Mankoff and Gregory D. Abowd. Cirrin: A word-level unistroke keyboard for pen input. In *ACM Symposium on User Interface Software and Technology*, pages 213–214. ACM Press, 1998.

[35] Merriam-Webster Inc. *Merriam-Webster Pocket Dictionary.* Merriam-Webster Inc., 1964. Computer readable form.

[36] Brad Myers, Jacob Wobbrock, Sunny Yang, Brian Yeung, Jeffrey Nichols, and Robert Miller. Using handhelds to help people with motor impairments. In *Proceedings of ASSETS 02*, pages 89–96. ACM Press, 2002.

[37] Oxford. *Oxford Dictionary of English.* Oxford University Press, 2004.

[38] Ken Perlin. Quikwriting: Continuous stylus-based text entry. In *ACM Symposium on User Interface Software and Technology*, pages 215–216, November 1998.

[39] Réjean Plamondon and Sargur N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 22, pages 63–84, January 2000.

[40] Nicholas J. Radcliffe. *Genetic neural networks on MIMD computers.* PhD thesis, Edinburgh, Scotland, UK, 1990.

[41] Colin R. Reeves. A genetic algorithm for flowshop sequencing. *Comput. Oper. Res.*, 22(1):5–13, 1995.

[42] Neil Rhodes and Julie McKeehan. *Palm OS Programming.* O'Reilly and Associates, 2nd edition, October 2001.

[43] Jennie Borodko Stack. *Palm Pilot Connects Girl with Classroom*, volume 8(1). Magazine of the Muscular Dystrophy Association, http://www.mdausa.org/publications/Quest/q81palmpilot.cfm, 2001.

[44] Tal Steinherz, Ehud Rivlin, and Nathon Intrator. Offline cursive script word recognition–a survey. *International Journal on Document Analysis and Recognition*, 2:90–110, 1999.

[45] Ching Suen, Marc Berthod, and Shunji Mori. Automatic recognition of handprinted characters – the state of the art. In *Proceedings of the IEEE*, volume 68, pages 469–487, April 1980.

[46] Charles Tappert. Speed, accuracy, and flexibility trade-offs in on-line character recognition. Technical Report RC13228, IBM Research, October 1987.

[47] Charles Tappert, Ching Suen, and Toru Wakahara. The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, August 1990.

[48] Dan Venolia and Forrest Neiberg. T-cube: A fast, self-disclosing pen-based alphabet. In *Proceedings of CHI Human Factors in Computing Systems*, pages 265–270. ACM Press, April 1994.

[49] Jacob Wobbrock, Brad Myers, and John Kembel. Edgewrite: A stylus-based text entry method designed for high accuracy and stability of motion. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '03)*, pages 61–70, November 2003.

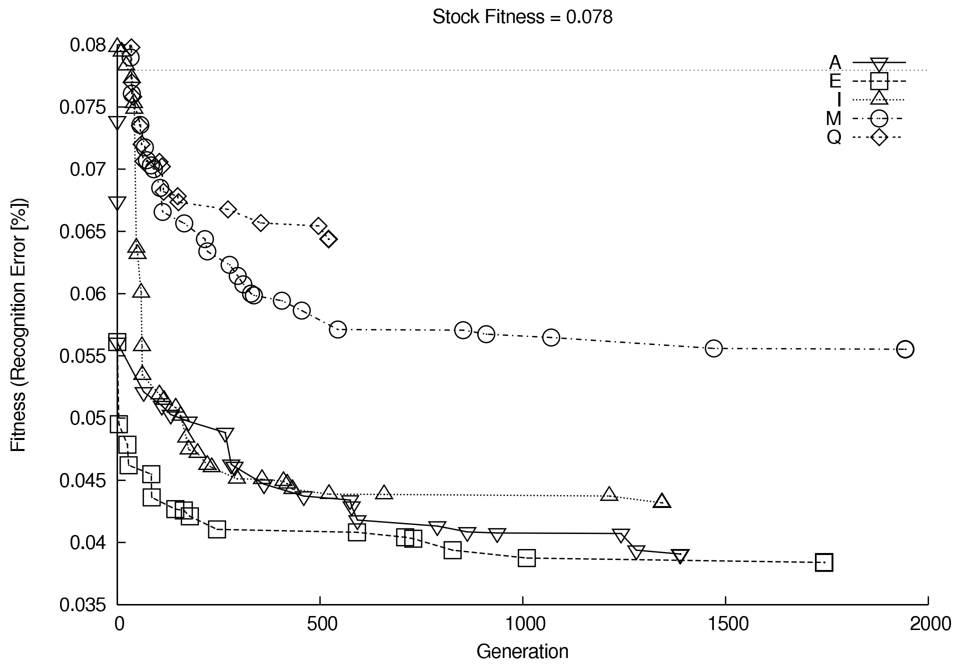
Appendices

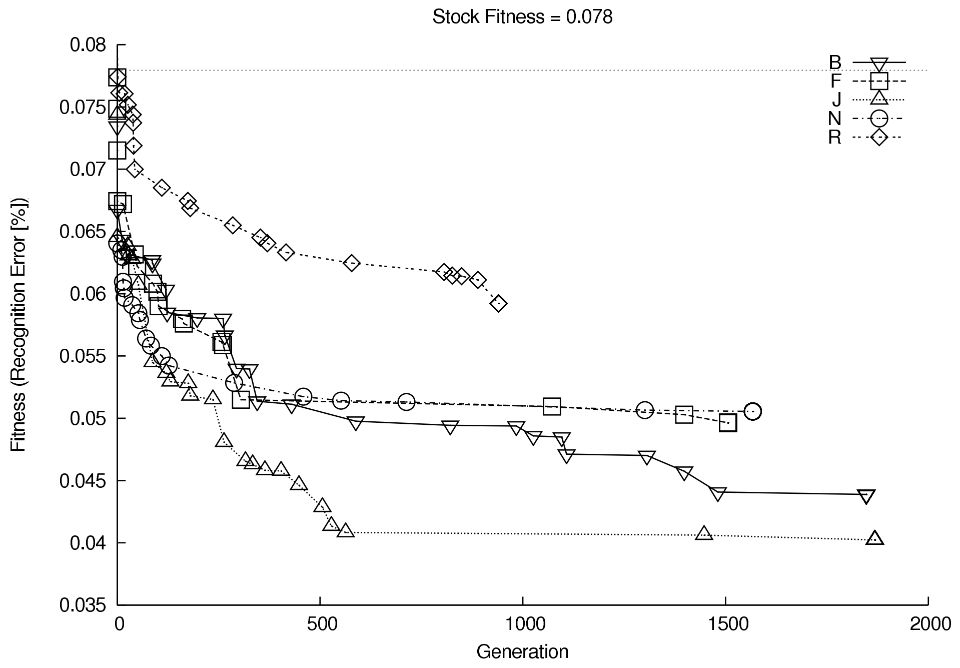
Appendix A

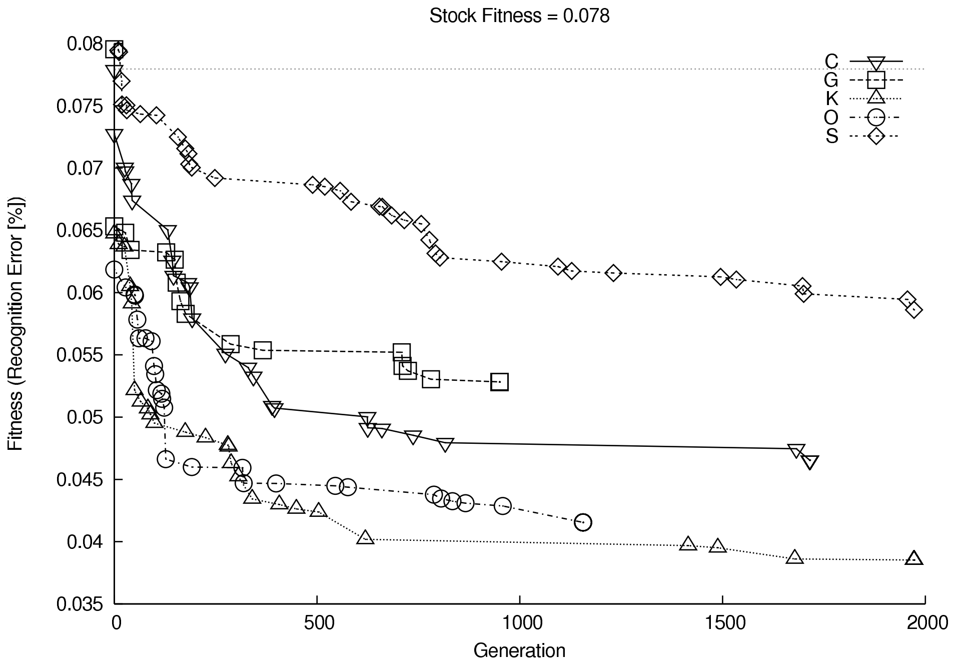
Genetic Algorithm Profiles

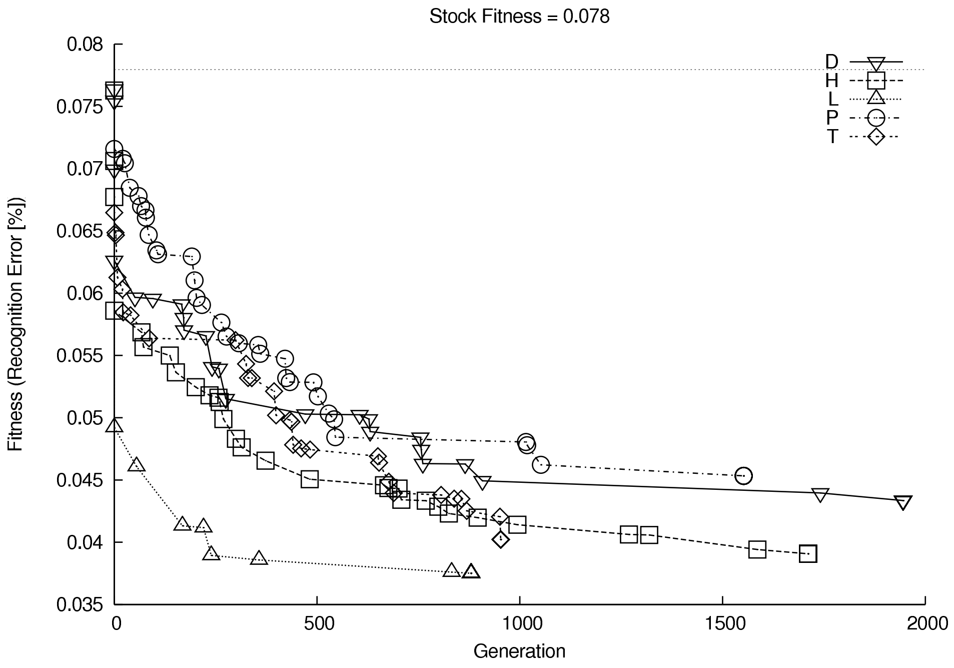
The following figures show the evolutionary progress of the 20 GA profiles defined in Section 6.3 for subjects “c00” and “c02”. For the sake of visualization clarity, runs associated with a particular subject and letter case combination have been distributed across four diagrams containing five profiles each. The top of each figure provides the fitness value when the stock parameter set for activity-based recognition was used with $\alpha = 1$. Further, a dotted horizontal line indicates this value in the figure. Each point along a particular run indicates when a new best solution was discovered.

Subject “c00” Profile Runs for Upper Case Characters

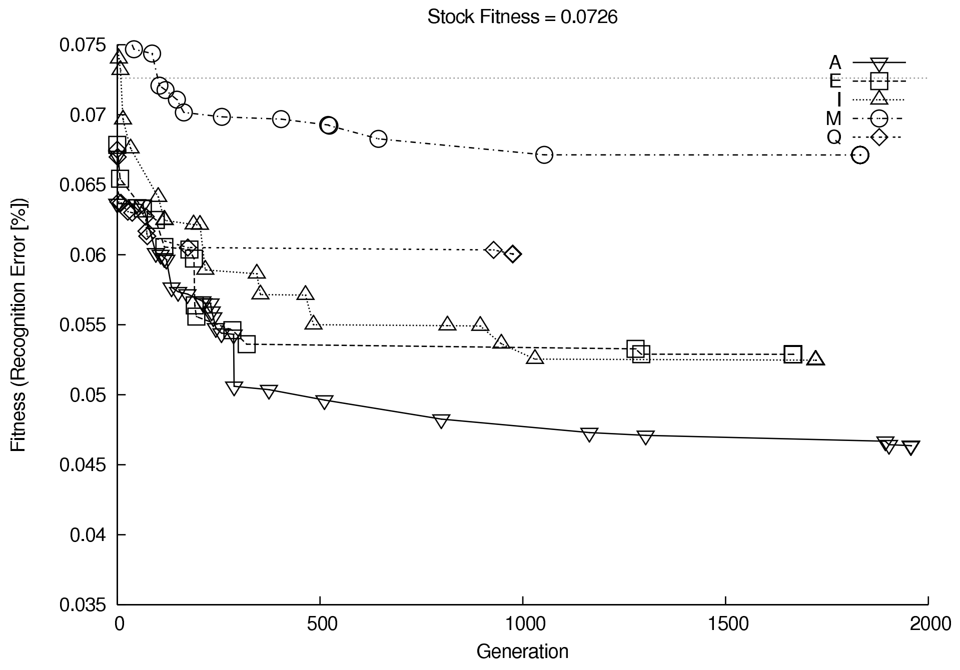


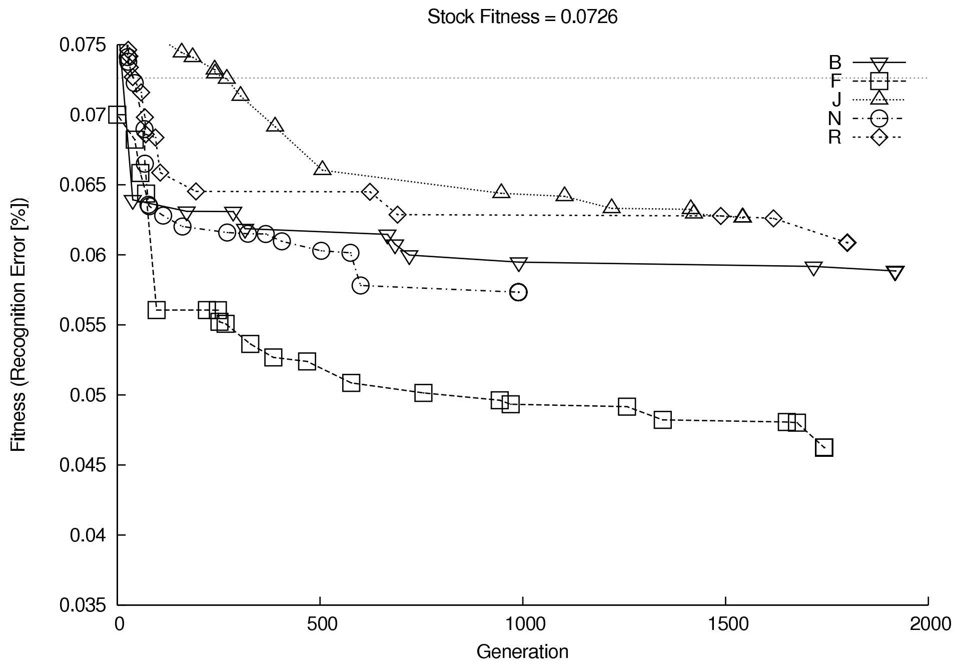


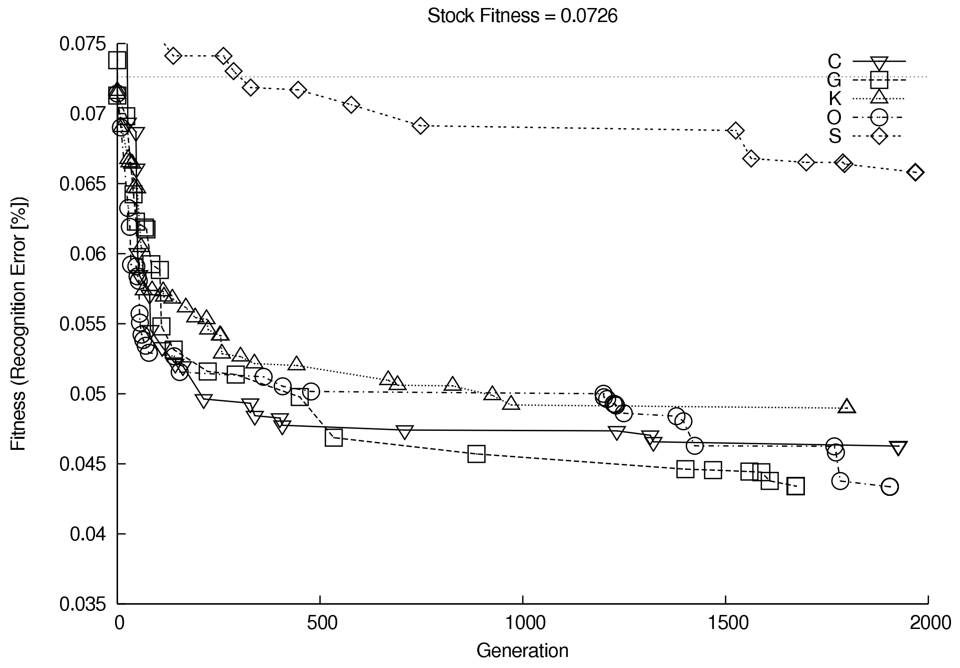

Stock Fitness = 0.078
Fitness (Recognition Error [%])
Generation
C
G
K
O
S
0.08
0.075
0.07
0.065
0.06
0.055
0.05
0.045
0.04
0.035
0
500
1000
1500
2000

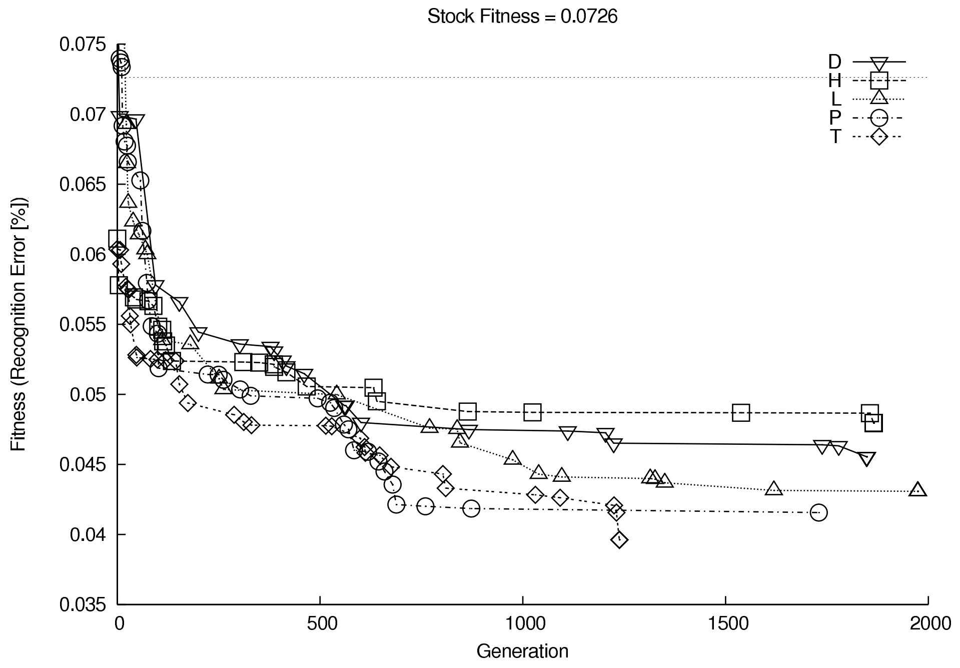

Stock Fitness = 0.078
Fitness (Recognition Error [%])
Generation
D
H
L
P
T
0.08
0.075
0.07
0.065
0.06
0.055
0.05
0.045
0.04
0.035
0
500
1000
1500
2000

Subject "c00" Profile Runs for Lower Case Characters

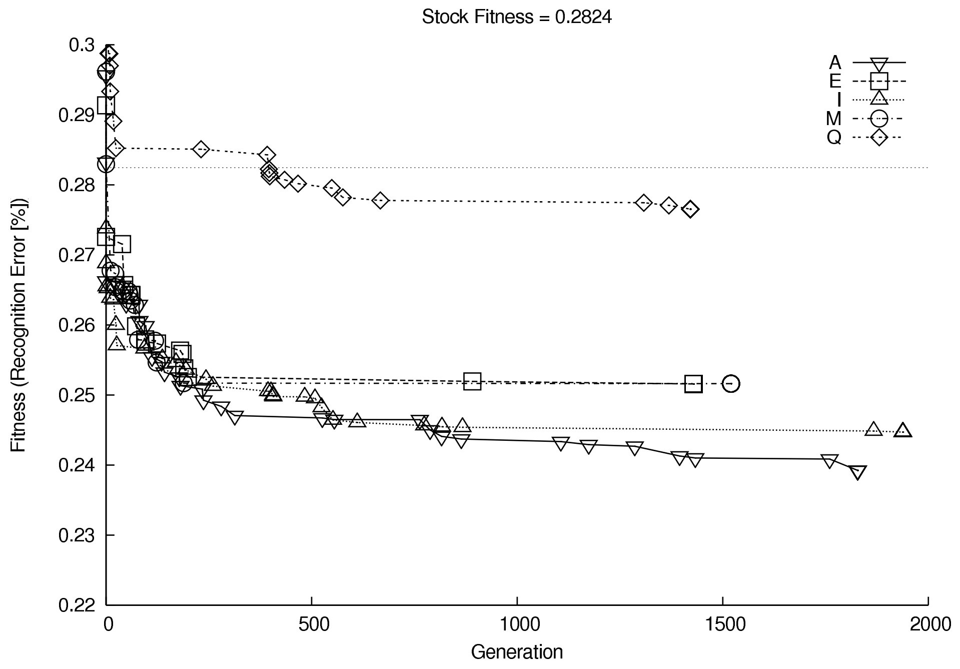


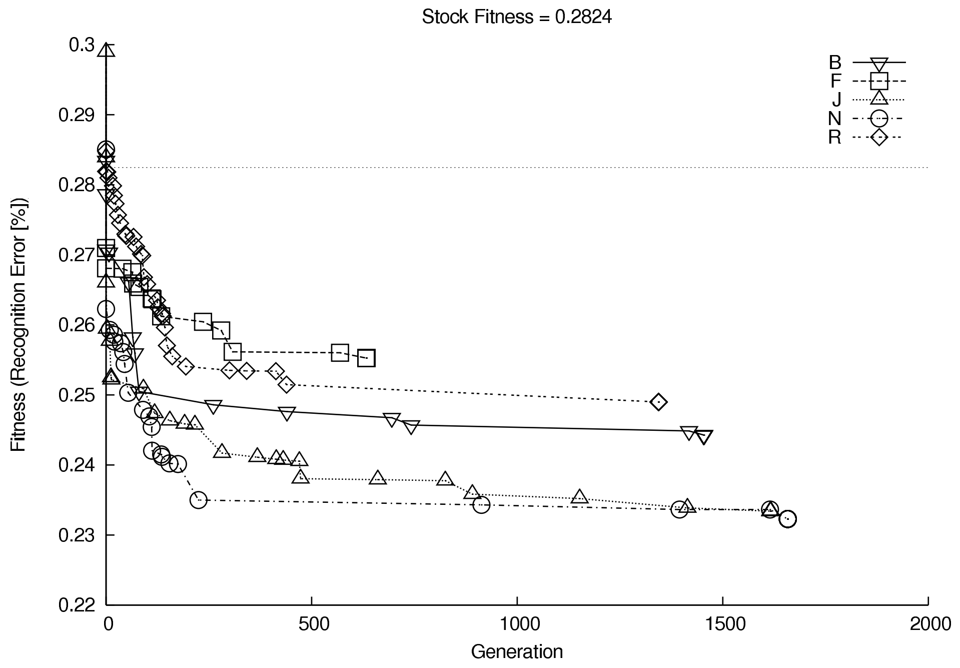


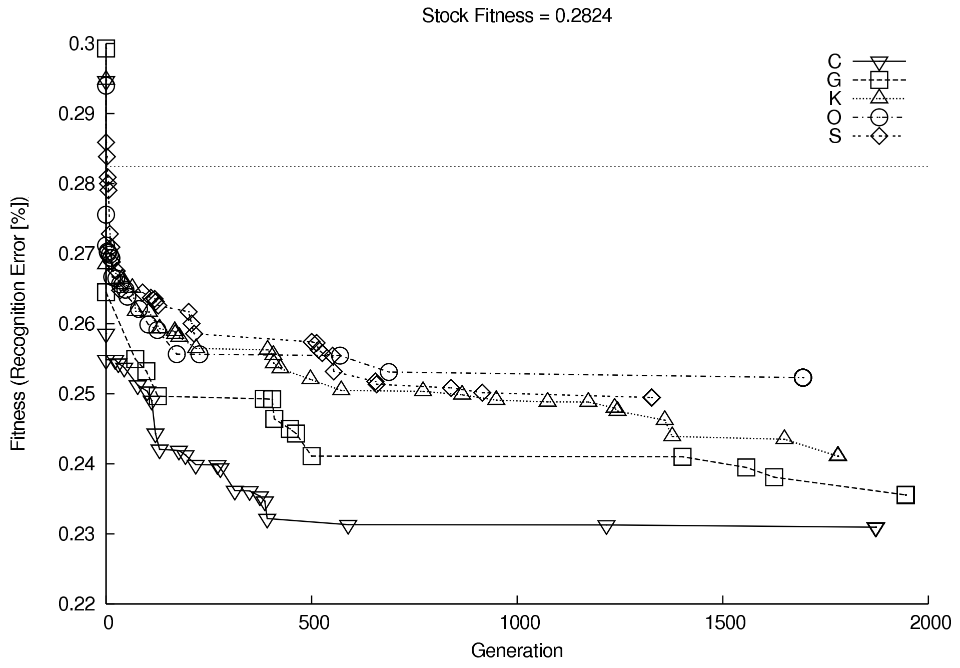

Stock Fitness = 0.0726
Fitness (Recognition Error [%])
Generation
C
G
K
O
S
0.075
0.07
0.065
0.06
0.055
0.05
0.045
0.04
0.035
0
500
1000
1500
2000

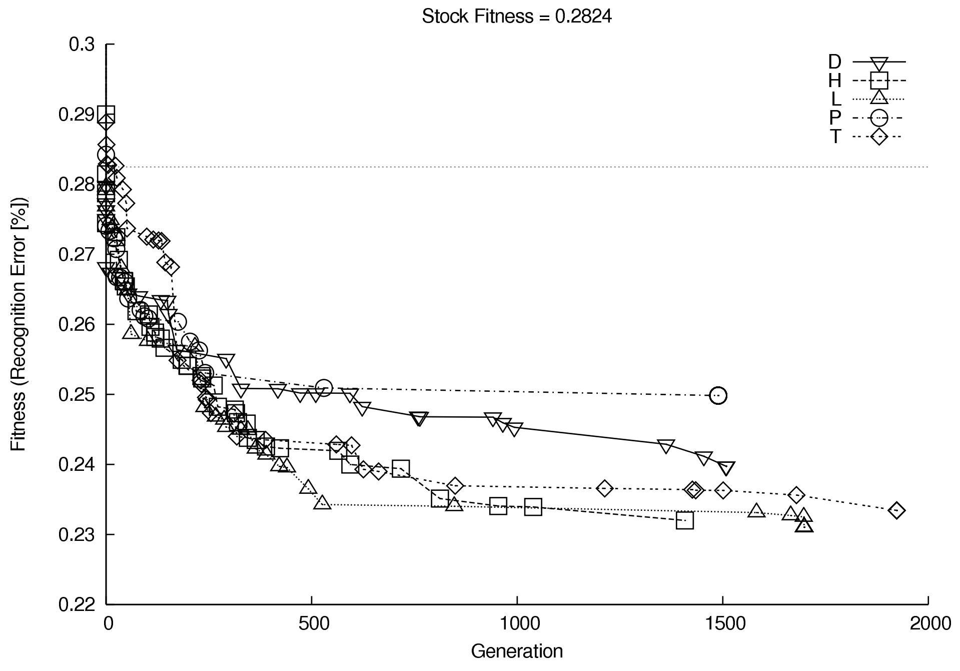

Stock Fitness = 0.0726
Fitness (Recognition Error [%])
Generation
D
H
L
P
T
0.075
0.07
0.065
0.06
0.055
0.05
0.045
0.04
0.035
0
500
1000
1500
2000

Subject "c02" Profile Runs for Upper Case Characters

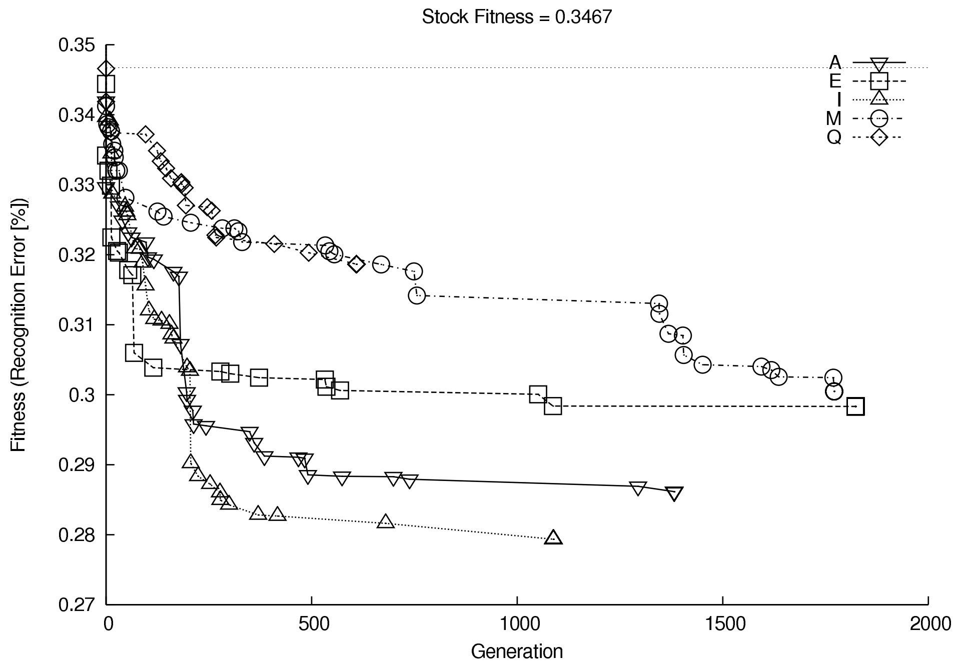


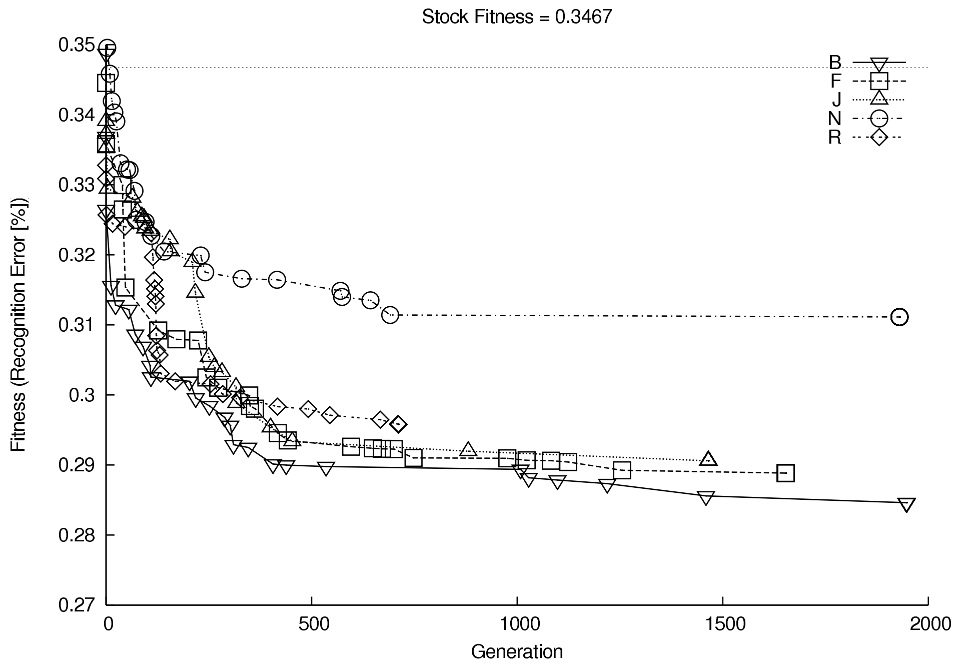


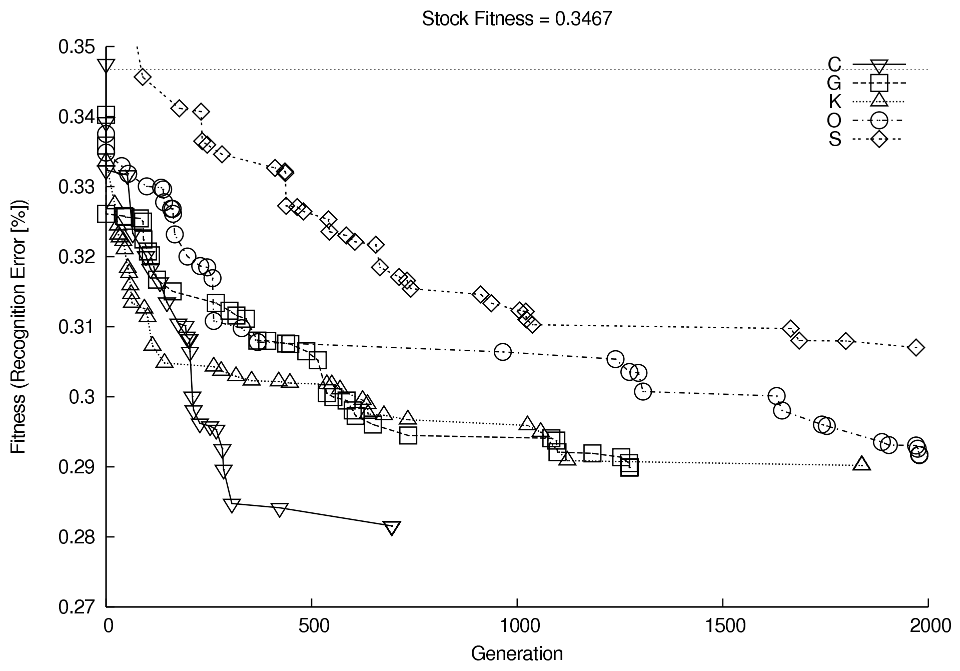

Stock Fitness = 0.2824
Fitness (Recognition Error [%])
Generation
C
G
K
O
S
0.3
0.29
0.28
0.27
0.26
0.25
0.24
0.23
0.22
0
500
1000
1500
2000

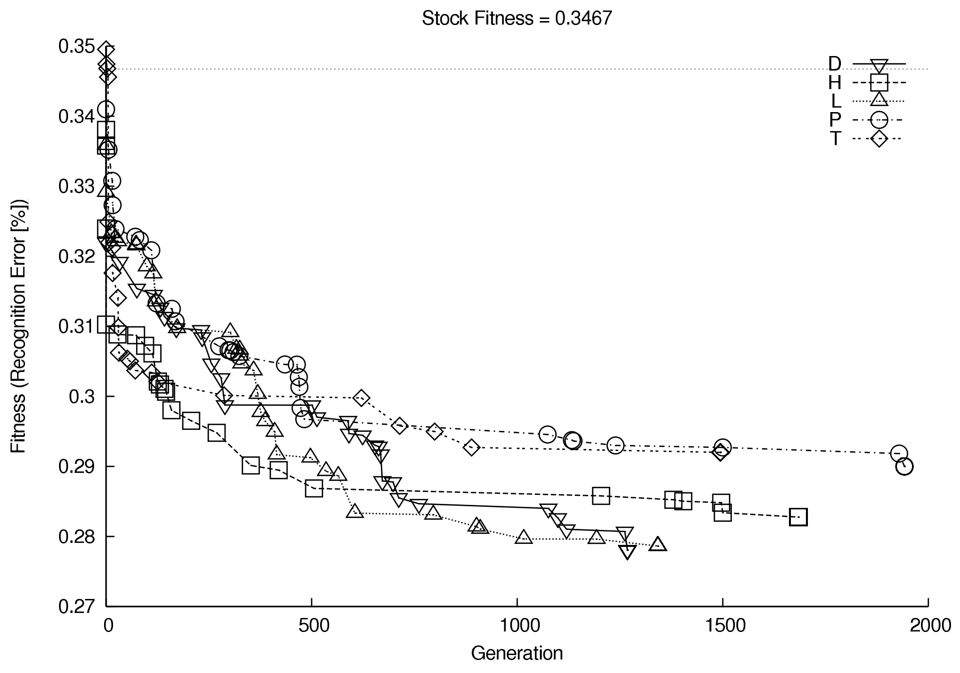

Stock Fitness = 0.2824
Fitness (Recognition Error [%])
Generation
D
H
L
P
T
0.3
0.29
0.28
0.27
0.26
0.25
0.24
0.23
0.22
0
500
1000
1500
2000

Subject “c02” Profile Runs for Lower Case Characters






Stock Fitness = 0.3467
Fitness (Recognition Error [%])
Generation
C
G
K
O
S
0.35
0.34
0.33
0.32
0.31
0.3
0.29
0.28
0.27
0
500
1000
1500
2000


Stock Fitness = 0.3467
Fitness (Recognition Error [%])
Generation
D
H
L
P
T
0.35
0.34
0.33
0.32
0.31
0.3
0.29
0.28
0.27
0
500
1000
1500
2000

Appendix B

Optimized Parameter Sets

The following figures represent the final parameters found in the optimization study described in Section 6.3. Each of the 66 subjects' upper and lower case sets are shown.

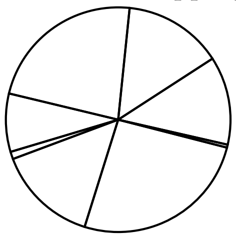
The figures contain four primary sections: error, directional mapping, activity regions, and scalar bias. The value labled "Error" indicates the percentage of characters misrecognized over the 300 randomly selected alphabets with $\alpha = 1$. The "Directional Mapping" shows the directional regions evolved. The directions are not labled 0–7 as with Freeman's chain code because they are inconsequential and their relative locations may have been extremely displaced during optimization. The "Activity Regions" portion of the figure identifies the starting and ending elements of the 32 resampled subtrokes of characters. The regions' relative size and position are visualized to the right of their respective values. The "Scalar Bias" portion of the figure identifies the scalar bias applied to the activity region visualized directly to its left.

Subject “c00”

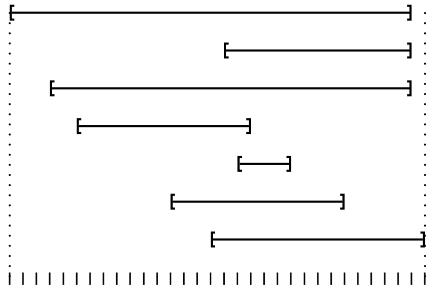
Optimized parameters for the upper case characters:

Error:
0.0410016

Directional Mapping:



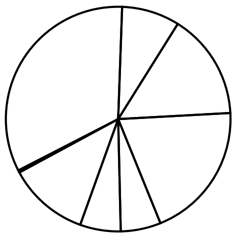
Activity Regions:	Scalar Bias:
[0,30]	185.537
[16,30]	17.7145
[3,30]	11.1813
[5,18]	21.4447
[17,21]	51.8673
[12,25]	136.92
[15,31]	142.06



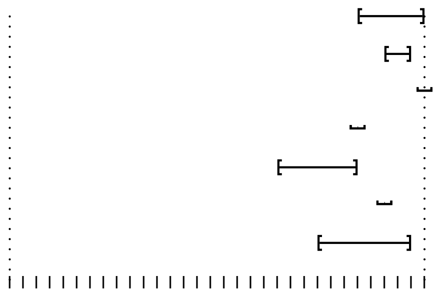
Optimized parameters for the lower case characters:

Error:
0.0443029

Directional Mapping:



Activity Regions:	Scalar Bias:
[26,31]	55.0411
[28,30]	34.0693
[31,31]	87.5962
[26,26]	166.645
[20,26]	71.4395
[28,28]	117.701
[23,30]	100.562

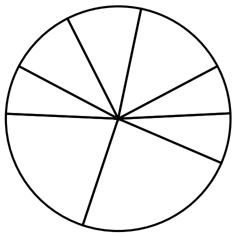


Subject “c01”

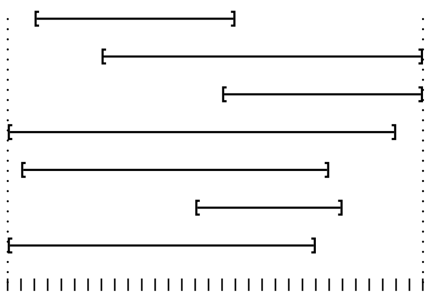
Optimized parameters for the upper case characters:

Error:
0.058101

Directional Mapping:



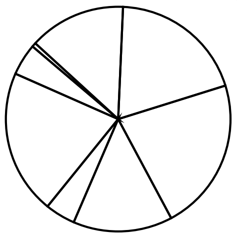
Activity Regions:	Scalar Bias:
[2,17]	136.426
[7,31]	63.3899
[16,31]	181.077
[0,29]	150.57
[1,24]	2.79018
[14,25]	35.8326
[0,23]	142.199



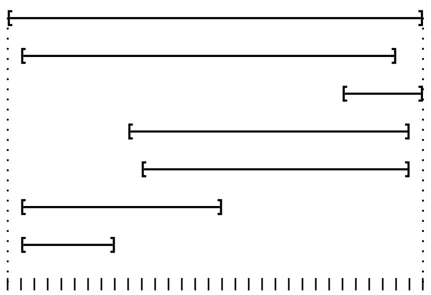
Optimized parameters for the lower case characters:

Error:
0.0511899

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,31]	157.718
[1,29]	6.90533
[25,31]	119.433
[9,30]	120.092
[10,30]	64.1443
[1,16]	137.248
[1,8]	105.585

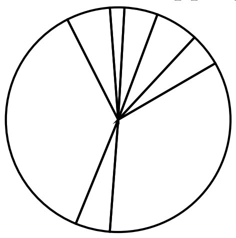


Subject "c02"

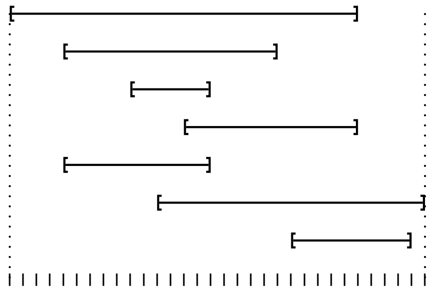
Optimized parameters for the upper case characters:

Error:
0.230966

Directional Mapping:



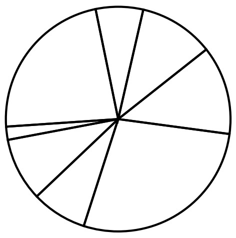
Activity Regions:	Scalar Bias:
[0,26]	162.876
[4,20]	87.7076
[9,15]	93.0607
[13,26]	108.461
[4,15]	43.5768
[11,31]	101.085
[21,30]	147.618



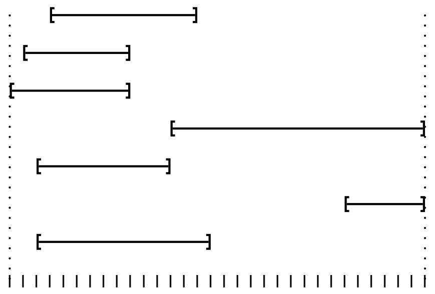
Optimized parameters for the lower case characters:

Error:
0.303401

Directional Mapping:



Activity Regions:	Scalar Bias:
[3,14]	11.0249
[1,9]	83.5426
[0,9]	51.254
[12,31]	108.478
[2,12]	138.925
[25,31]	192.959
[2,15]	22.9986

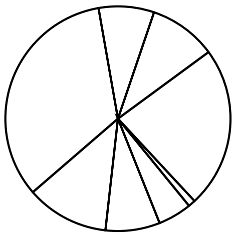


Subject “c03”

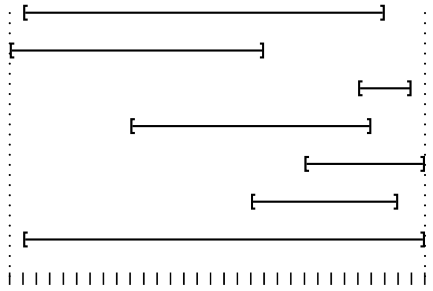
Optimized parameters for the upper case characters:

Error:
0.051238

Directional Mapping:



Activity Regions:

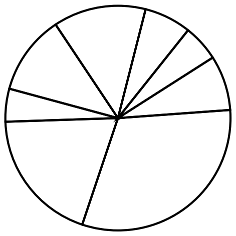


Activity Regions:	Scalar Bias:
[1,28]	112.255
[0,19]	155.743
[26,30]	169.523
[9,27]	74.1164
[22,31]	14.7265
[18,29]	105.092
[1,31]	67.3367

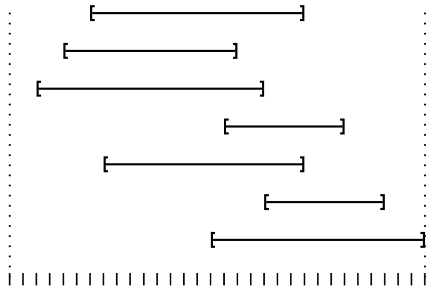
Optimized parameters for the lower case characters:

Error:
0.029992

Directional Mapping:



Activity Regions:



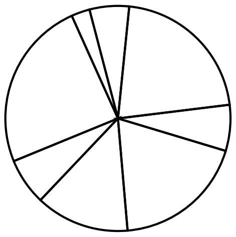
Activity Regions:	Scalar Bias:
[6,22]	14.6319
[4,17]	156.118
[2,19]	63.18
[16,25]	94.4849
[7,22]	203.149
[19,28]	90.5955
[15,31]	144.082

Subject “c04”

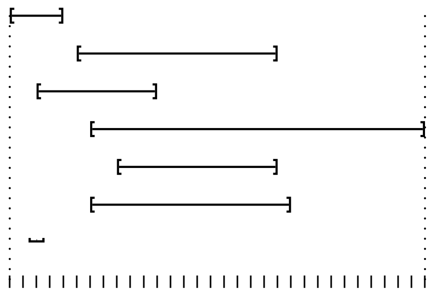
Optimized parameters for the upper case characters:

Error:
0.155116

Directional Mapping:



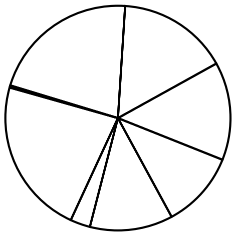
Activity Regions:	Scalar Bias:
[0,4]	17.3706
[5,20]	8.91946
[2,11]	112.537
[6,31]	128.533
[8,20]	56.0872
[6,21]	121.381
[2,2]	179.17



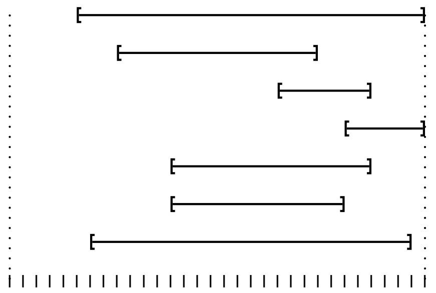
Optimized parameters for the lower case characters:

Error:
0.100717

Directional Mapping:



Activity Regions:	Scalar Bias:
[5,31]	158.82
[8,23]	109.861
[20,27]	81.8627
[25,31]	134.747
[12,27]	29.0503
[12,25]	45.4129
[6,30]	43.0312

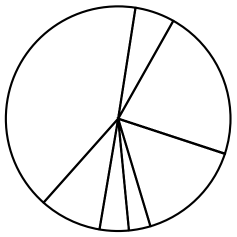


Subject “c05”

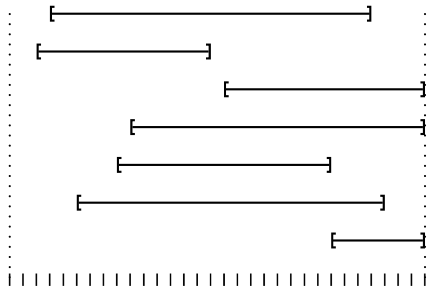
Optimized parameters for the upper case characters:

Error:
0.0775521

Directional Mapping:



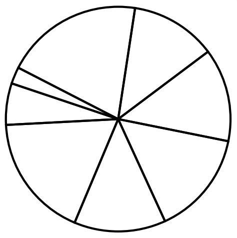
Activity Regions:	Scalar Bias:
[3,27]	99.3617
[2,15]	122.451
[16,31]	163.135
[9,31]	133.347
[8,24]	109.603
[5,28]	23.6801
[24,31]	87.3497



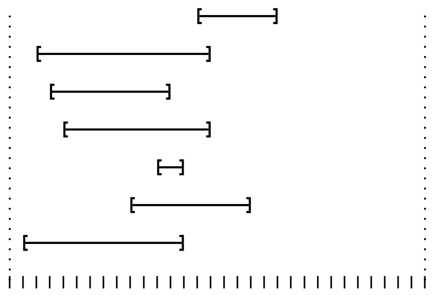
Optimized parameters for the lower case characters:

Error:
0.0288301

Directional Mapping:



Activity Regions:	Scalar Bias:
[14,20]	55.3274
[2,15]	65.7895
[3,12]	104.774
[4,15]	81.6605
[11,13]	30.7791
[9,18]	155.566
[1,13]	15.0562

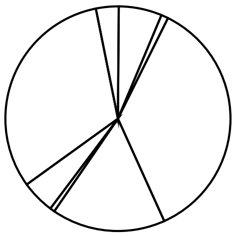


Subject “c06”

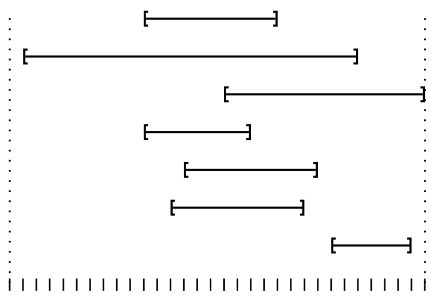
Optimized parameters for the upper case characters:

Error:
0.0741066

Directional Mapping:



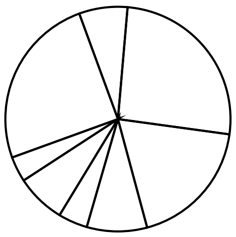
Activity Regions:	Scalar Bias:
[10,20]	82.8961
[1,26]	141.647
[16,31]	179.869
[10,18]	22.3728
[13,23]	71.9964
[12,22]	100.111
[24,30]	147.686



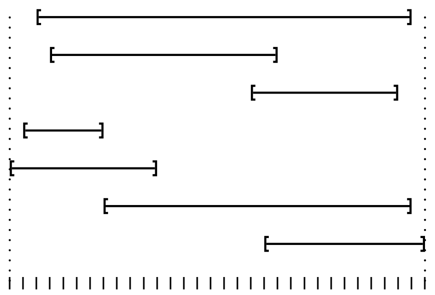
Optimized parameters for the lower case characters:

Error:
0.160084

Directional Mapping:



Activity Regions:	Scalar Bias:
[2,30]	195.456
[3,20]	77.921
[18,29]	101.737
[1,7]	108.11
[0,11]	18.0115
[7,30]	3.52932
[19,31]	61.4518

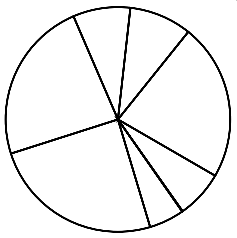


Subject “c07”

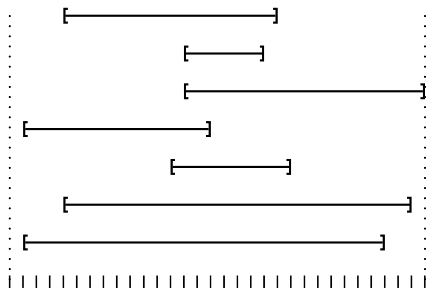
Optimized parameters for the upper case characters:

Error:
0.1098

Directional Mapping:



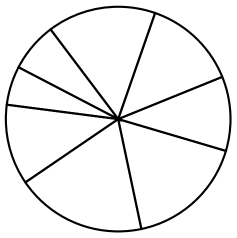
Activity Regions:	Scalar Bias:
[4,20]	71.3732
[13,19]	52.3073
[13,31]	123.79
[1,15]	149.695
[12,21]	127.072
[4,30]	181.333
[1,28]	67.194



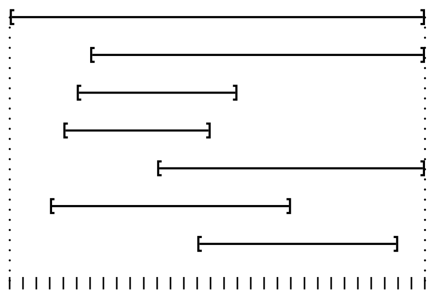
Optimized parameters for the lower case characters:

Error:
0.125084

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,31]	138.889
[6,31]	42.0388
[5,17]	47.6082
[4,15]	35.8665
[11,31]	85.6894
[3,21]	106
[14,29]	13.2575

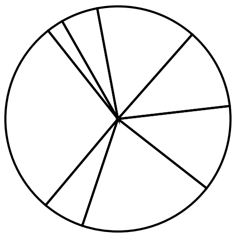


Subject “c08”

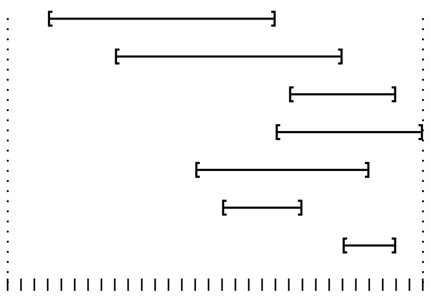
Optimized parameters for the upper case characters:

Error:
0.225284

Directional Mapping:



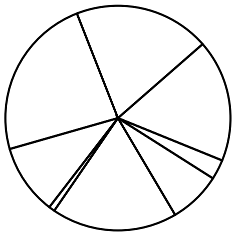
Activity Regions:	Scalar Bias:
[3,20]	85.1883
[8,25]	80.1829
[21,29]	100.548
[20,31]	117.299
[14,27]	46.1111
[16,22]	79.6613
[25,29]	104.756



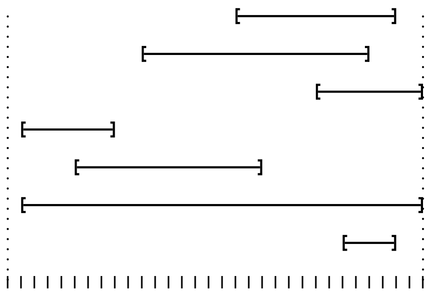
Optimized parameters for the lower case characters:

Error:
0.180737

Directional Mapping:



Activity Regions:	Scalar Bias:
[17,29]	75.6946
[10,27]	92.0391
[23,31]	154.623
[1,8]	79.9464
[5,19]	138.443
[1,31]	145.356
[25,29]	77.5635

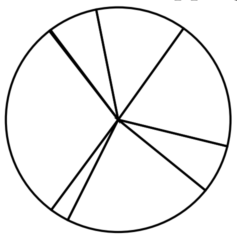


Subject "c09"

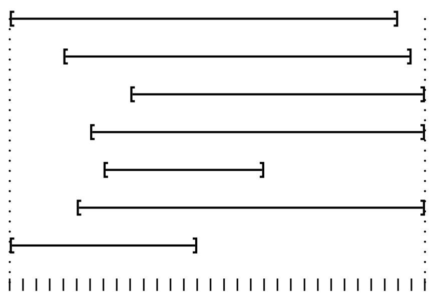
Optimized parameters for the upper case characters:

Error:
0.0330288

Directional Mapping:



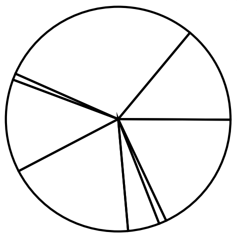
Activity Regions:	Scalar Bias:
[0,29]	110.891
[4,30]	70.2981
[9,31]	115.82
[6,31]	77.2957
[7,19]	178.596
[5,31]	29.4995
[0,14]	88.878



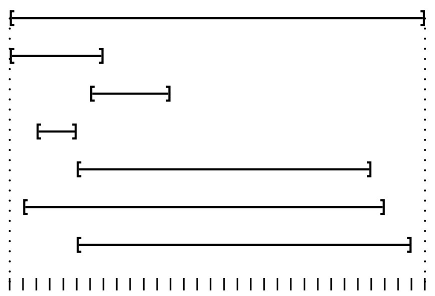
Optimized parameters for the lower case characters:

Error:
0.0615425

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,31]	58.508
[0,7]	76.71
[6,12]	61.0807
[2,5]	90.1035
[5,27]	111.117
[1,28]	41.402
[5,30]	187.038

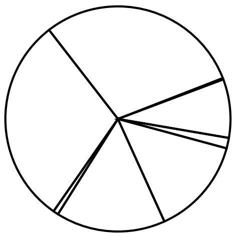


Subject “c10”

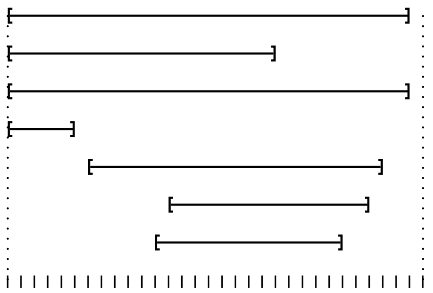
Optimized parameters for the upper case characters:

Error:
0.096226

Directional Mapping:



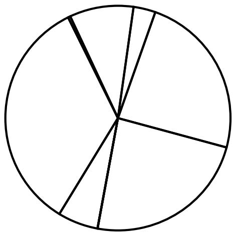
Activity Regions:	Scalar Bias:
[0,30]	127.517
[0,20]	174.062
[0,30]	3.58045
[0,5]	85.5529
[6,28]	77.2626
[12,27]	175.321
[11,25]	24.0055



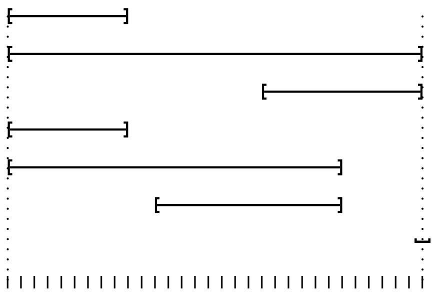
Optimized parameters for the lower case characters:

Error:
0.116146

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,9]	71.9252
[0,31]	51.8543
[19,31]	103.3
[0,9]	8.32579
[0,25]	154.298
[11,25]	167.56
[31,31]	163.94

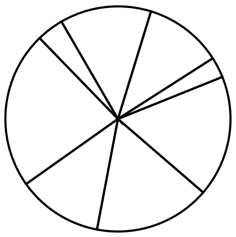


Subject “c11”

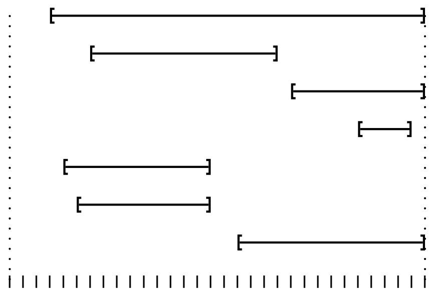
Optimized parameters for the upper case characters:

Error:
0.352432

Directional Mapping:



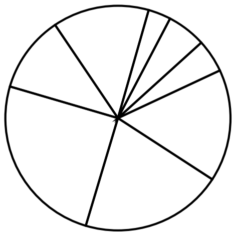
Activity Regions:	Scalar Bias:
[3,31]	120.639
[6,20]	105.957
[21,31]	112.357
[26,30]	98.811
[4,15]	6.2912
[5,15]	67.0133
[17,31]	80.5539



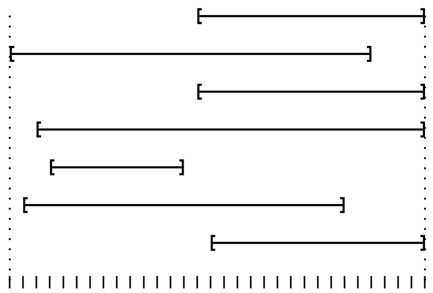
Optimized parameters for the lower case characters:

Error:
0.368221

Directional Mapping:



Activity Regions:	Scalar Bias:
[14,31]	33.8572
[0,27]	85.1457
[14,31]	21.5791
[2,31]	86.8324
[3,13]	95.7103
[1,25]	124.911
[15,31]	121.399

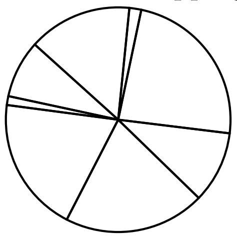


Subject “c12”

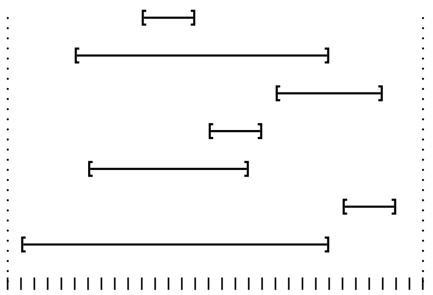
Optimized parameters for the upper case characters:

Error:
0.0852764

Directional Mapping:



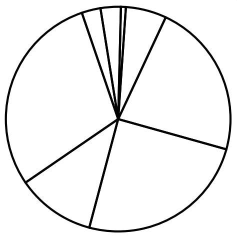
Activity Regions:	Scalar Bias:
[10,14]	45.2465
[5,24]	51.5572
[20,28]	148.809
[15,19]	39.5054
[6,18]	123.807
[25,29]	81.4921
[1,24]	122.678



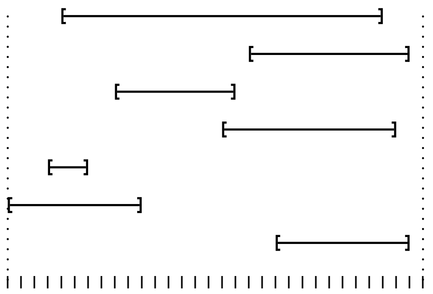
Optimized parameters for the lower case characters:

Error:
0.100393

Directional Mapping:



Activity Regions:	Scalar Bias:
[4,28]	194.055
[18,30]	160.358
[8,17]	115.052
[16,29]	36.4885
[3,6]	69.4201
[0,10]	82.8582
[20,30]	92.9649

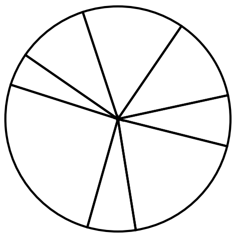


Subject "c13"

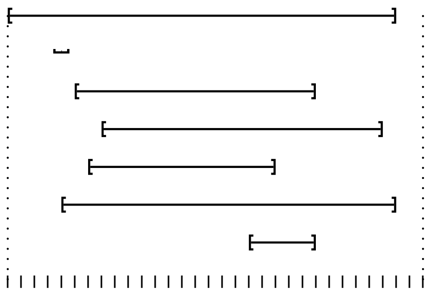
Optimized parameters for the upper case characters:

Error:
0.0584696

Directional Mapping:



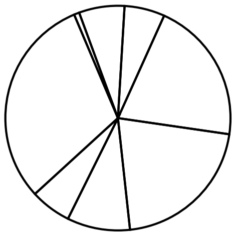
Activity Regions:	Scalar Bias:
[0,29]	126.546
[4,4]	163.278
[5,23]	128.427
[7,28]	122.223
[6,20]	71.1779
[4,29]	27.588
[18,23]	70.8643



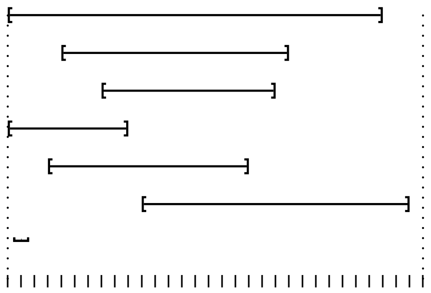
Optimized parameters for the lower case characters:

Error:
0.0817468

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,28]	91.3544
[4,21]	55.6963
[7,20]	51.4464
[0,9]	136.894
[3,18]	29.5846
[10,30]	126.187
[1,1]	191.123

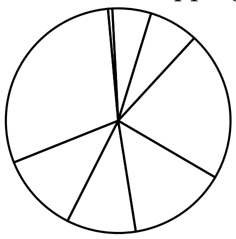


Subject “c14”

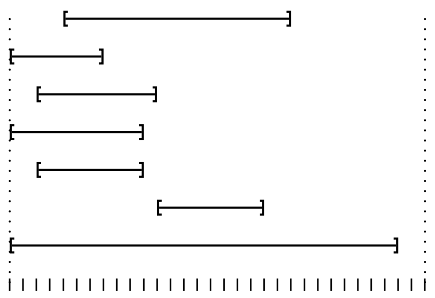
Optimized parameters for the upper case characters:

Error:
0.190232

Directional Mapping:



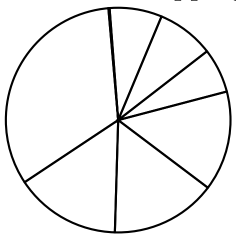
Activity Regions:	Scalar Bias:
[4,21]	74.574
[0,7]	4.63831
[2,11]	124.618
[0,10]	31.6933
[2,10]	27.5384
[11,19]	91.7298
[0,29]	95.9236



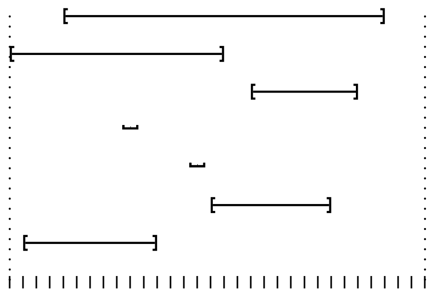
Optimized parameters for the lower case characters:

Error:
0.163045

Directional Mapping:



Activity Regions:	Scalar Bias:
[4,28]	124.307
[0,16]	97.4474
[18,26]	1.41646
[9,9]	105.94
[14,14]	2.49281
[15,24]	124.886
[1,11]	86.9893

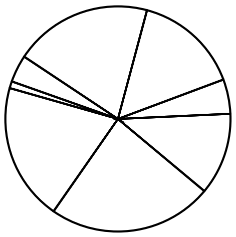


Subject “c15”

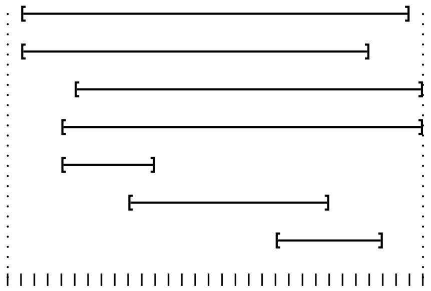
Optimized parameters for the upper case characters:

Error:
0.173229

Directional Mapping:



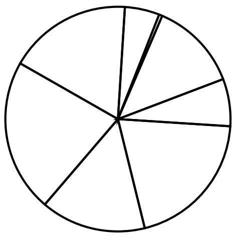
Activity Regions:	Scalar Bias:
[1,30]	131.346
[1,27]	117.624
[5,31]	123.601
[4,31]	57.027
[4,11]	198.104
[9,24]	127.319
[20,28]	180.028



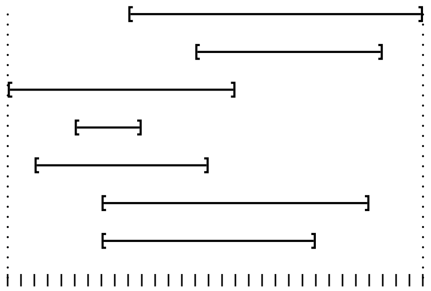
Optimized parameters for the lower case characters:

Error:
0.189339

Directional Mapping:



Activity Regions:	Scalar Bias:
[9,31]	132.592
[14,28]	66.293
[0,17]	105.454
[5,10]	43.0787
[2,15]	58.2831
[7,27]	4.68164
[7,23]	139.905

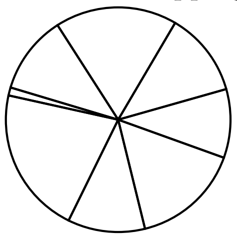


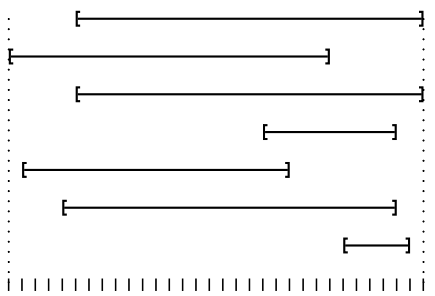
Subject "c16"

Optimized parameters for the upper case characters:

Error:
0.188474

Directional Mapping:



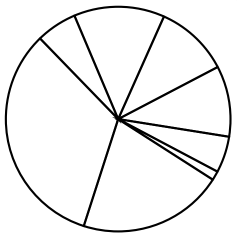


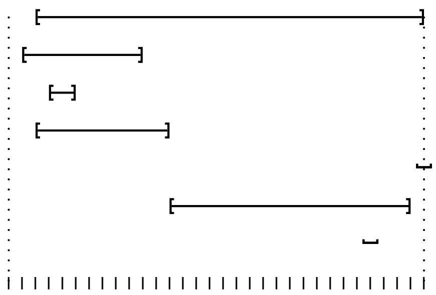
Activity Regions:	Scalar Bias:
[5,31]	133.249
[0,24]	7.40678
[5,31]	21.4927
[19,29]	67.096
[1,21]	145.932
[4,29]	13.6645
[25,30]	128.048

Optimized parameters for the lower case characters:

Error:
0.25246

Directional Mapping:





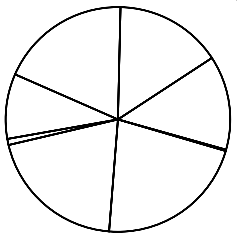
Activity Regions:	Scalar Bias:
[2,31]	161.8
[1,10]	134.076
[3,5]	31.618
[2,12]	52.5634
[31,31]	144.059
[12,30]	128.453
[27,27]	163.559

Subject "c17"

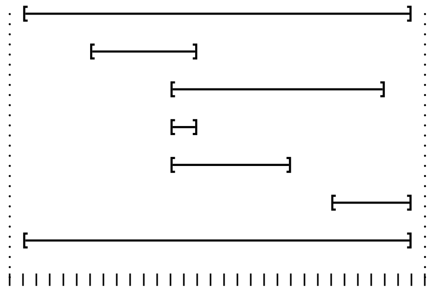
Optimized parameters for the upper case characters:

Error:
0.178377

Directional Mapping:



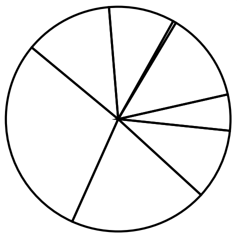
Activity Regions:	Scalar Bias:
[1,30]	112.74
[6,14]	177.555
[12,28]	114.96
[12,14]	15.7983
[12,21]	160.963
[24,30]	153.645
[1,30]	47.9266



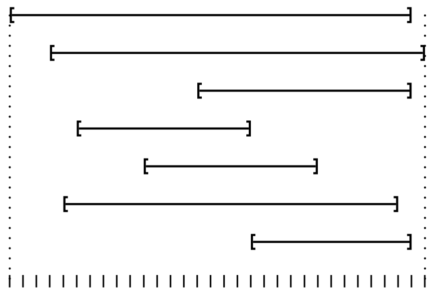
Optimized parameters for the lower case characters:

Error:
0.131747

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,30]	143.016
[3,31]	123.759
[14,30]	179.759
[5,18]	3.67423
[10,23]	169.625
[4,29]	63.1713
[18,30]	11.7781

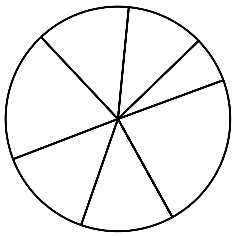


Subject “c18”

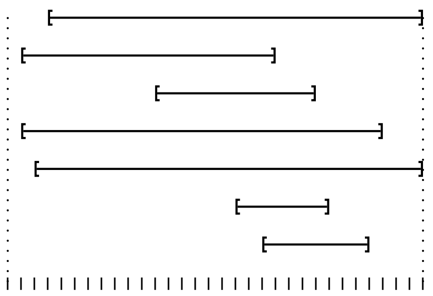
Optimized parameters for the upper case characters:

Error:
0.242957

Directional Mapping:



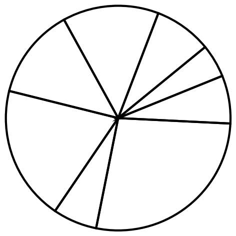
Activity Regions:	Scalar Bias:
[3,31]	12.7487
[1,20]	133.61
[11,23]	105.813
[1,28]	54.9401
[2,31]	29.293
[17,24]	112.439
[19,27]	132.182



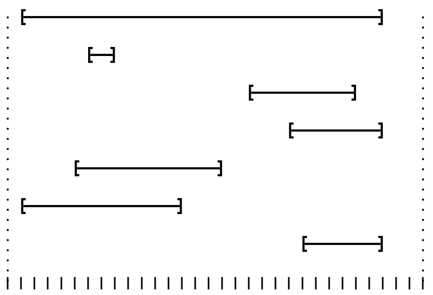
Optimized parameters for the lower case characters:

Error:
0.451651

Directional Mapping:



Activity Regions:	Scalar Bias:
[1,28]	128.763
[6,8]	52.4643
[18,26]	128.914
[21,28]	72.5034
[5,16]	96.0161
[1,13]	69.5062
[22,28]	117.787

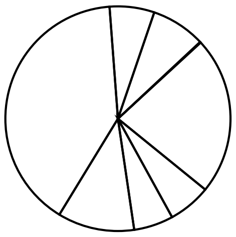


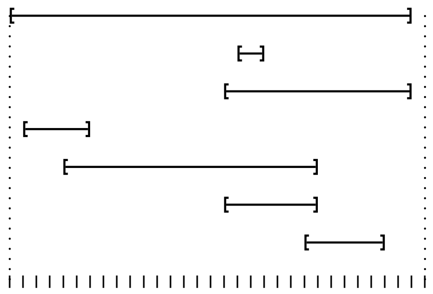
Subject "c19"

Optimized parameters for the upper case characters:

Error:
0.0910457

Directional Mapping:



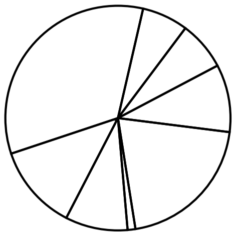


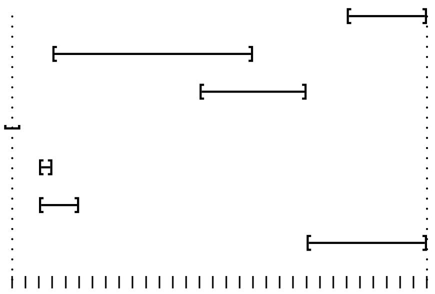
Activity Regions:	Scalar Bias:
[0,30]	117.977
[17,19]	0.704248
[16,30]	55.2683
[1,6]	147.879
[4,23]	195.228
[16,23]	73.226
[22,28]	123.516

Optimized parameters for the lower case characters:

Error:
0.10869

Directional Mapping:





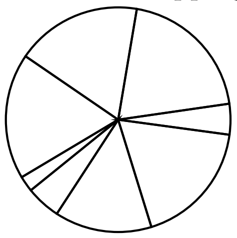
Activity Regions:	Scalar Bias:
[25,31]	153.385
[3,18]	133.468
[14,22]	154.669
[0,0]	133.217
[2,3]	78.9633
[2,5]	38.4741
[22,31]	99.6347

Subject “c20”

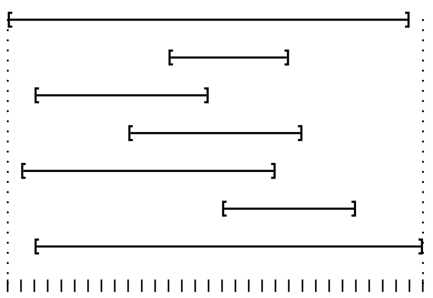
Optimized parameters for the upper case characters:

Error:
0.10401

Directional Mapping:



Activity Regions:

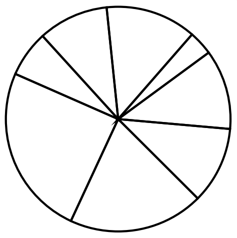


Activity Regions:	Scalar Bias:
[0,30]	70.0801
[12,21]	71.7994
[2,15]	97.9465
[9,22]	78.51
[1,20]	86.3814
[16,26]	79.0717
[2,31]	141.365

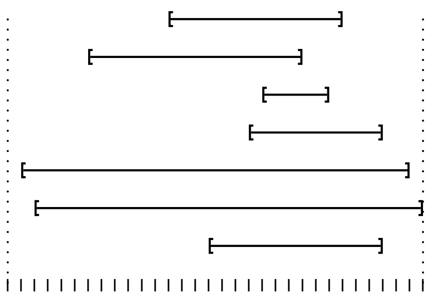
Optimized parameters for the lower case characters:

Error:
0.099976

Directional Mapping:



Activity Regions:



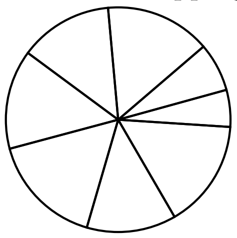
Activity Regions:	Scalar Bias:
[12,25]	91.9222
[6,22]	62.7307
[19,24]	69.567
[18,28]	42.4467
[1,30]	167.164
[2,31]	50.852
[15,28]	108.156

Subject “c21”

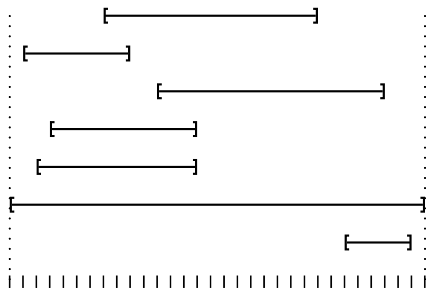
Optimized parameters for the upper case characters:

Error:
0.105938

Directional Mapping:



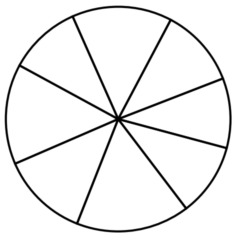
Activity Regions:	Scalar Bias:
[7,23]	104.64
[1,9]	107.769
[11,28]	104.577
[3,14]	88.2079
[2,14]	129.125
[0,31]	110.042
[25,30]	106.803



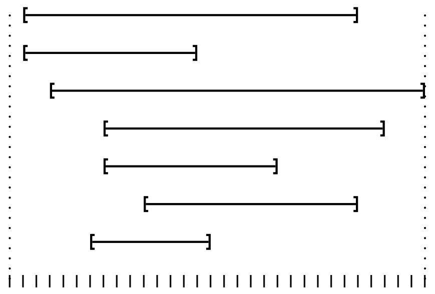
Optimized parameters for the lower case characters:

Error:
0.166987

Directional Mapping:



Activity Regions:	Scalar Bias:
[1,26]	19.6133
[1,14]	26.2741
[3,31]	117.532
[7,28]	3.51239
[7,20]	98.3833
[10,26]	87.4932
[6,15]	116.404

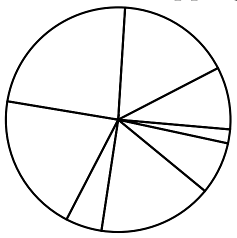


Subject “c22”

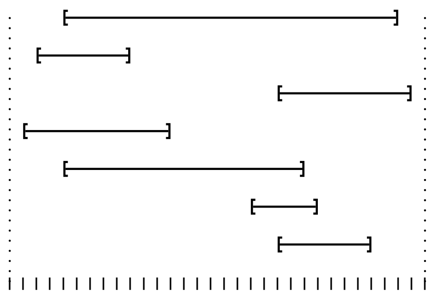
Optimized parameters for the upper case characters:

Error:
0.0985136

Directional Mapping:



Activity Regions:

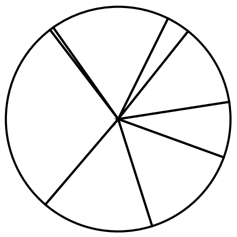


Activity Regions:	Scalar Bias:
[4,29]	100.463
[2,9]	9.69649
[20,30]	107.138
[1,12]	85.2906
[4,22]	40.2513
[18,23]	125.511
[20,27]	100.177

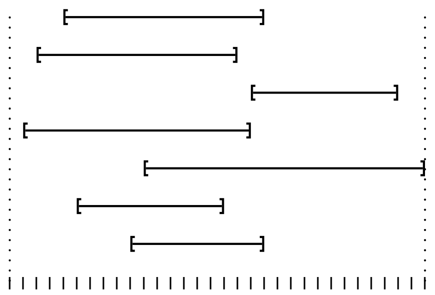
Optimized parameters for the lower case characters:

Error:
0.107973

Directional Mapping:



Activity Regions:



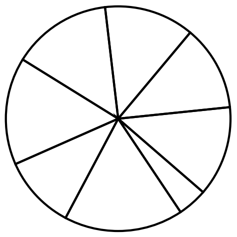
Activity Regions:	Scalar Bias:
[4,19]	95.805
[2,17]	55.2773
[18,29]	0.767765
[1,18]	22.0513
[10,31]	144.078
[5,16]	8.42714
[9,19]	71.296

Subject “c23”

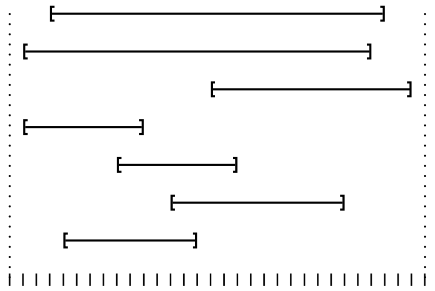
Optimized parameters for the upper case characters:

Error:
0.0459255

Directional Mapping:



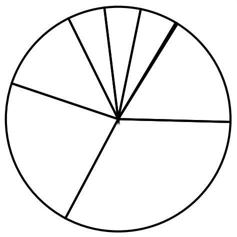
Activity Regions:	Scalar Bias:
[3,28]	166.535
[1,27]	48.1232
[15,30]	54.2834
[1,10]	139.658
[8,17]	157.928
[12,25]	115.305
[4,14]	142.843



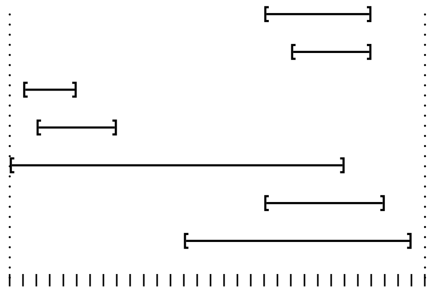
Optimized parameters for the lower case characters:

Error:
0.158421

Directional Mapping:



Activity Regions:	Scalar Bias:
[19,27]	26.9372
[21,27]	46.5957
[1,5]	14.3676
[2,8]	169.508
[0,25]	167.415
[19,28]	59.2531
[13,30]	188.806

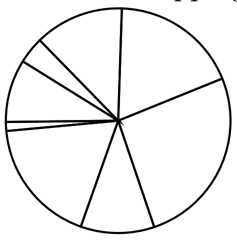


Subject “c24”

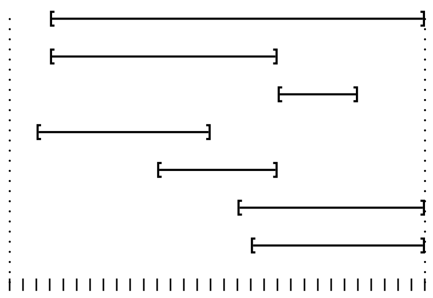
Optimized parameters for the upper case characters:

Error:
0.128337

Directional Mapping:



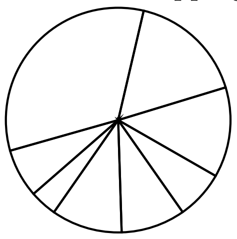
Activity Regions:	Scalar Bias:
[3,31]	103.784
[3,20]	158.971
[20,26]	7.05251
[2,15]	79.0556
[11,20]	113.236
[17,31]	105.349
[18,31]	120.853



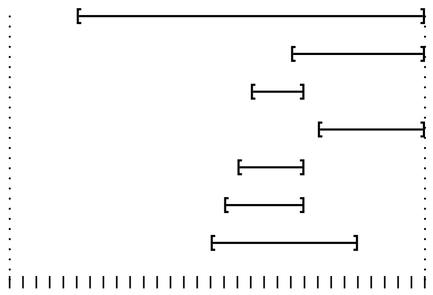
Optimized parameters for the lower case characters:

Error:
0.230697

Directional Mapping:



Activity Regions:	Scalar Bias:
[5,31]	84.5366
[21,31]	35.5753
[18,22]	30.7789
[23,31]	170.502
[17,22]	3.06102
[16,22]	30.2358
[15,26]	85.5438

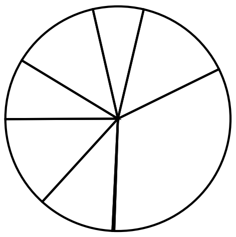


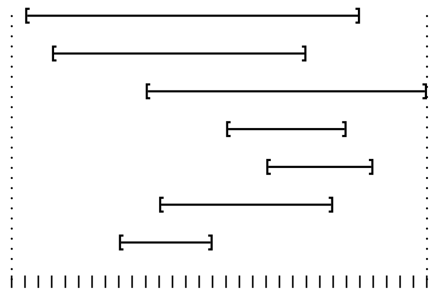
Subject “c25”

Optimized parameters for the upper case characters:

Error:
0.110441

Directional Mapping:



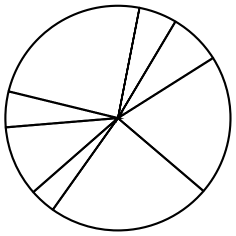


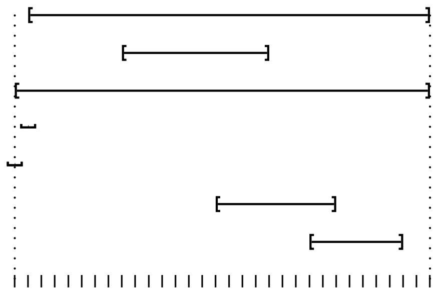
Activity Regions:	Scalar Bias:
[1,26]	131.113
[3,22]	47.1443
[10,31]	142.653
[16,25]	166.562
[19,27]	88.3611
[11,24]	102.986
[8,15]	86.9191

Optimized parameters for the lower case characters:

Error:
0.0878005

Directional Mapping:





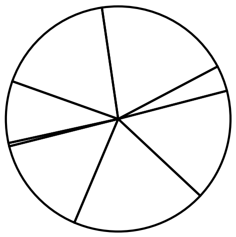
Activity Regions:	Scalar Bias:
[1,31]	174.107
[8,19]	117.828
[0,31]	77.2542
[1,1]	187.011
[0,0]	169.715
[15,24]	94.3359
[22,29]	95.9883

Subject “c26”

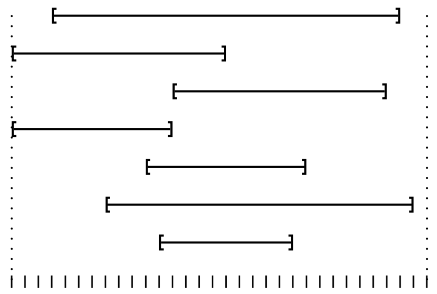
Optimized parameters for the upper case characters:

Error:
0.158393

Directional Mapping:



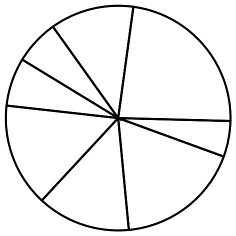
Activity Regions:	Scalar Bias:
[3,29]	77.9367
[0,16]	133.889
[12,28]	26.6941
[0,12]	48.9754
[10,22]	132.367
[7,30]	67.233
[11,21]	20.5276



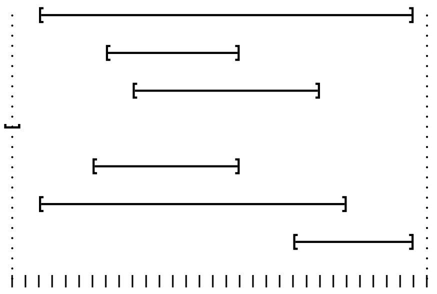
Optimized parameters for the lower case characters:

Error:
0.125605

Directional Mapping:



Activity Regions:	Scalar Bias:
[2,30]	102.317
[7,17]	58.5418
[9,23]	33.984
[0,0]	141.842
[6,17]	84.8043
[2,25]	44.9699
[21,30]	133.235

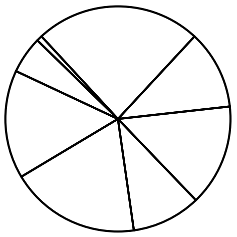


Subject “c27”

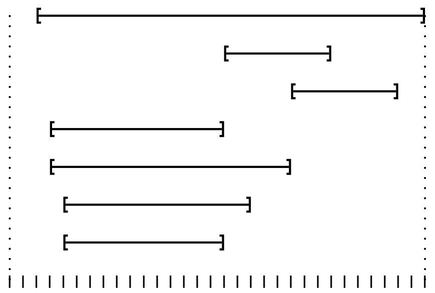
Optimized parameters for the upper case characters:

Error:
0.138269

Directional Mapping:



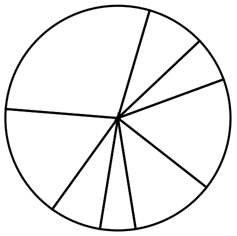
Activity Regions:	Scalar Bias:
[2,31]	115.837
[16,24]	110.802
[21,29]	161.646
[3,16]	77.2869
[3,21]	74.1725
[4,18]	30.9143
[4,16]	36.1901



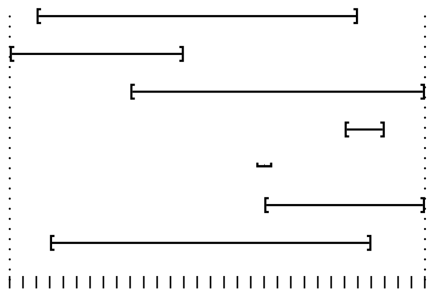
Optimized parameters for the lower case characters:

Error:
0.21899

Directional Mapping:



Activity Regions:	Scalar Bias:
[2,26]	94.7931
[0,13]	81.91
[9,31]	86.9897
[25,28]	35.1895
[19,19]	48.6636
[19,31]	176.465
[3,27]	16.5838

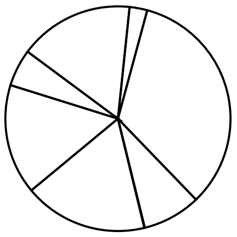


Subject "c28"

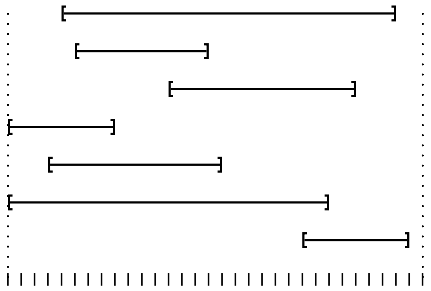
Optimized parameters for the upper case characters:

Error:
0.0530449

Directional Mapping:



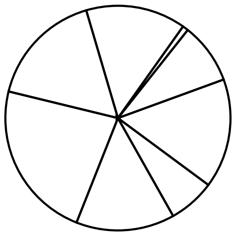
Activity Regions:	Scalar Bias:
[4,29]	49.6022
[5,15]	154.69
[12,26]	139.311
[0,8]	90.2627
[3,16]	141.394
[0,24]	55.9702
[22,30]	138.516



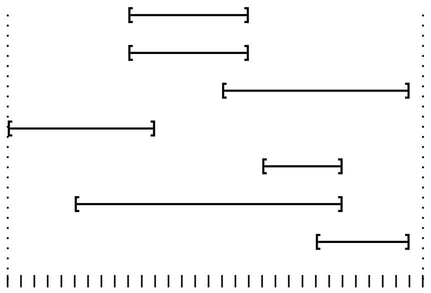
Optimized parameters for the lower case characters:

Error:
0.0738782

Directional Mapping:



Activity Regions:	Scalar Bias:
[9,18]	37.7973
[9,18]	109.265
[16,30]	95.6369
[0,11]	81.1552
[19,25]	35.3687
[5,25]	142.531
[23,30]	128.703

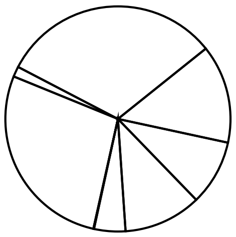


Subject “c29”

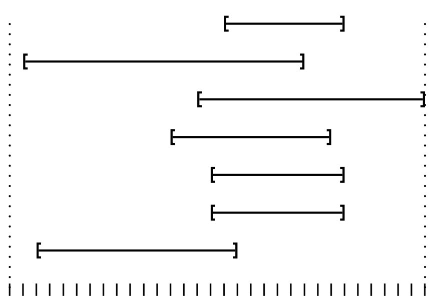
Optimized parameters for the upper case characters:

Error:
0.0439223

Directional Mapping:



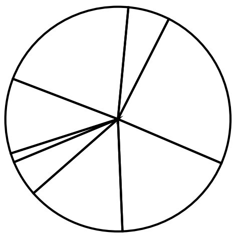
Activity Regions:	Scalar Bias:
[16,25]	49.69
[1,22]	159.729
[14,31]	108.55
[12,24]	44.629
[15,25]	106.144
[15,25]	38.9062
[2,17]	41.95



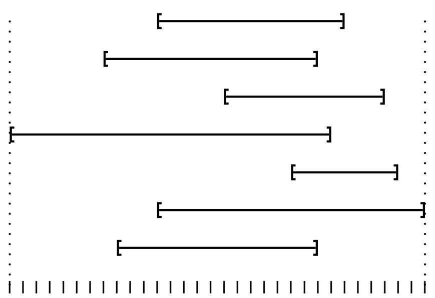
Optimized parameters for the lower case characters:

Error:
0.0557171

Directional Mapping:



Activity Regions:	Scalar Bias:
[11,25]	4.62448
[7,23]	84.0787
[16,28]	36.8133
[0,24]	152.515
[21,29]	102.208
[11,31]	12.5449
[8,23]	26.9113

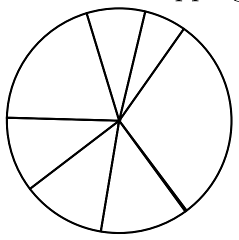


Subject “c30”

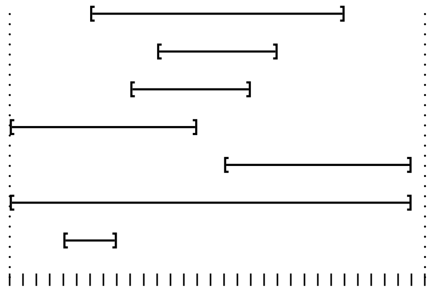
Optimized parameters for the upper case characters:

Error:
0.17387

Directional Mapping:



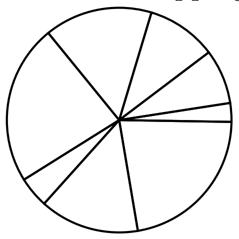
Activity Regions:	Scalar Bias:
[6,25]	101.921
[11,20]	127.514
[9,18]	22.1682
[0,14]	98.5478
[16,30]	91.3762
[0,30]	101.325
[4,8]	13.7437



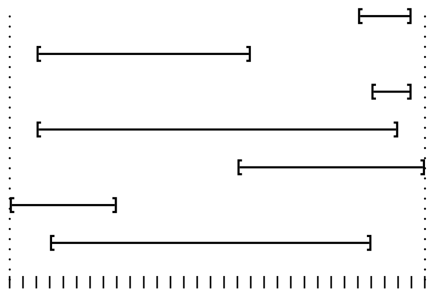
Optimized parameters for the lower case characters:

Error:
0.200845

Directional Mapping:



Activity Regions:	Scalar Bias:
[26,30]	35.8078
[2,18]	92.5925
[27,30]	5.0718
[2,29]	88.711
[17,31]	140.547
[0,8]	167.198
[3,27]	6.25048

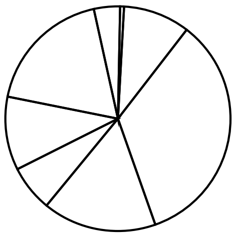


Subject “c31”

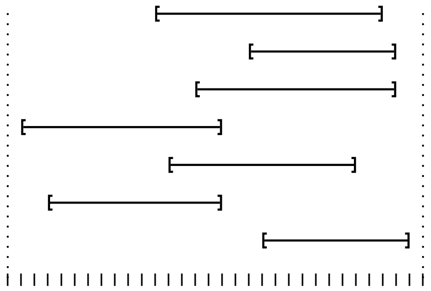
Optimized parameters for the upper case characters:

Error:
0.120757

Directional Mapping:



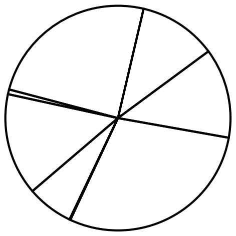
Activity Regions:	Scalar Bias:
[11,28]	159.779
[18,29]	110.287
[14,29]	82.7398
[1,16]	95.1861
[12,26]	13.2275
[3,16]	36.6346
[19,30]	47.9338



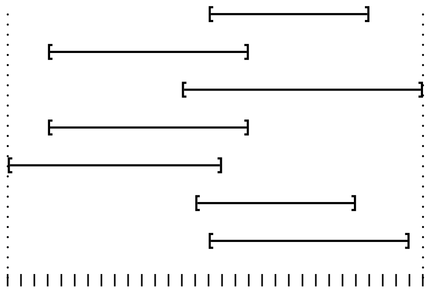
Optimized parameters for the lower case characters:

Error:
0.0832893

Directional Mapping:



Activity Regions:	Scalar Bias:
[15,27]	21.7192
[3,18]	59.2801
[13,31]	186.036
[3,18]	19.6122
[0,16]	85.3168
[14,26]	60.787
[15,30]	170.976

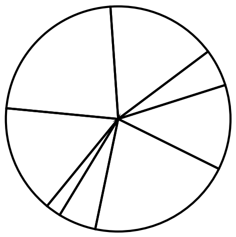


Subject "c32"

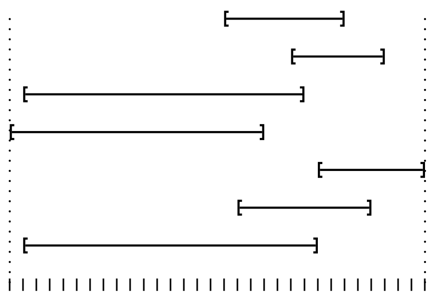
Optimized parameters for the upper case characters:

Error:
0.147716

Directional Mapping:



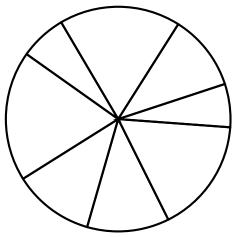
Activity Regions:	Scalar Bias:
[16,25]	22.6267
[21,28]	73.2952
[1,22]	54.8834
[0,19]	58.9387
[23,31]	96.1215
[17,27]	51.0723
[1,23]	13.3136



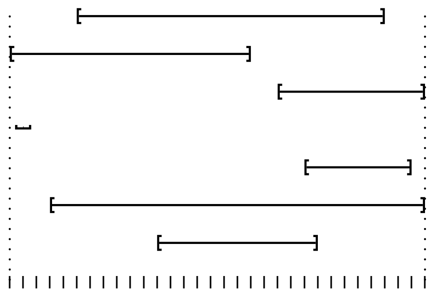
Optimized parameters for the lower case characters:

Error:
0.180184

Directional Mapping:



Activity Regions:	Scalar Bias:
[5,28]	42.3307
[0,18]	71.1163
[20,31]	73.9211
[1,1]	164.327
[22,30]	64.556
[3,31]	51.2183
[11,23]	9.20665

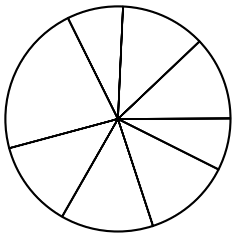


Subject "c33"

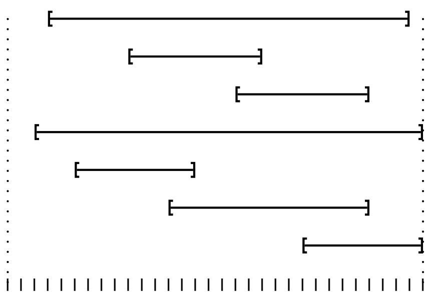
Optimized parameters for the upper case characters:

Error:
0.159471

Directional Mapping:



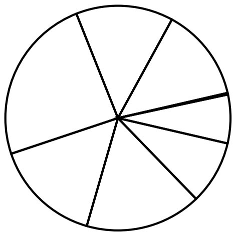
Activity Regions:	Scalar Bias:
[3,30]	136.797
[9,19]	146.854
[17,27]	68.4648
[2,31]	101.091
[5,14]	175.395
[12,27]	118.705
[22,31]	196.08



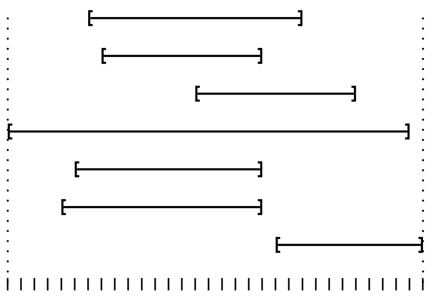
Optimized parameters for the lower case characters:

Error:
0.161146

Directional Mapping:



Activity Regions:	Scalar Bias:
[6,22]	37.525
[7,19]	83.033
[14,26]	134.255
[0,30]	175.849
[5,19]	84.8813
[4,19]	0.167313
[20,31]	177.888

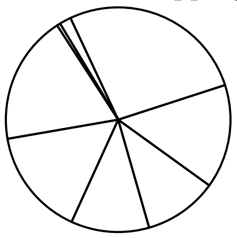


Subject “c34”

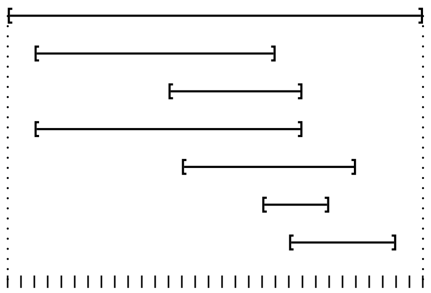
Optimized parameters for the upper case characters:

Error:
0.152135

Directional Mapping:



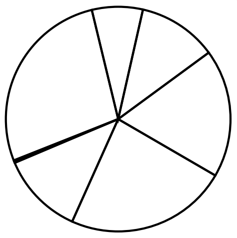
Activity Regions:	Scalar Bias:
[0,31]	141.577
[2,20]	70.7026
[12,22]	21.8886
[2,22]	82.3364
[13,26]	124.539
[19,24]	22.2652
[21,29]	144.256



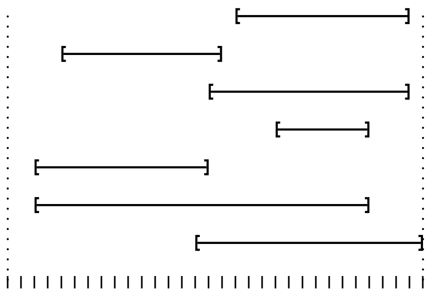
Optimized parameters for the lower case characters:

Error:
0.154563

Directional Mapping:



Activity Regions:	Scalar Bias:
[17,30]	34.7362
[4,16]	116.774
[15,30]	170.166
[20,27]	22.5639
[2,15]	52.48
[2,27]	172.387
[14,31]	79.9799

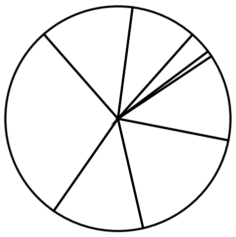


Subject "c35"

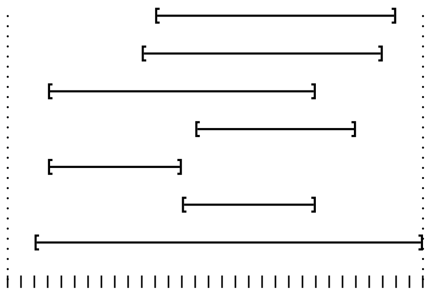
Optimized parameters for the upper case characters:

Error:
0.065028

Directional Mapping:



Activity Regions:

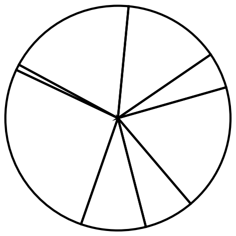


Activity Regions:	Scalar Bias:
[11,29]	3.53714
[10,28]	93.9488
[3,23]	76.4152
[14,26]	50.4936
[3,13]	65.8857
[13,23]	91.4465
[2,31]	104.08

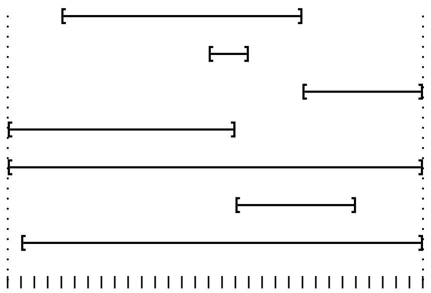
Optimized parameters for the lower case characters:

Error:
0.105276

Directional Mapping:



Activity Regions:



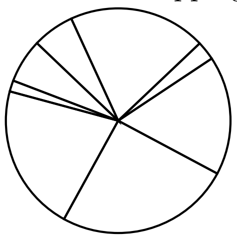
Activity Regions:	Scalar Bias:
[4,22]	91.5004
[15,18]	47.8863
[22,31]	93.0178
[0,17]	4.25931
[0,31]	125.151
[17,26]	107.225
[1,31]	68.7355

Subject “c36”

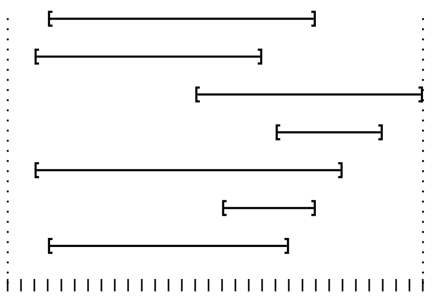
Optimized parameters for the upper case characters:

Error:
0.0795593

Directional Mapping:



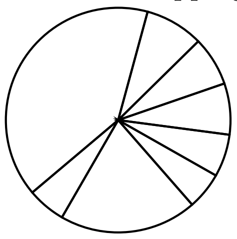
Activity Regions:	Scalar Bias:
[3,23]	61.8527
[2,19]	186.245
[14,31]	189.072
[20,28]	76.4317
[2,25]	41.7045
[16,23]	164.246
[3,21]	56.174



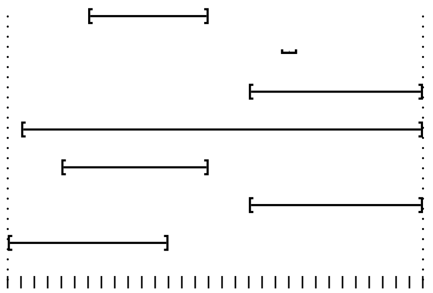
Optimized parameters for the lower case characters:

Error:
0.0763221

Directional Mapping:



Activity Regions:	Scalar Bias:
[6,15]	12.2069
[21,21]	159.612
[18,31]	158.056
[1,31]	109.423
[4,15]	87.0328
[18,31]	24.4837
[0,12]	55.4194

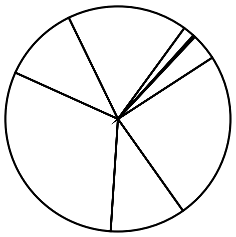


Subject “c37”

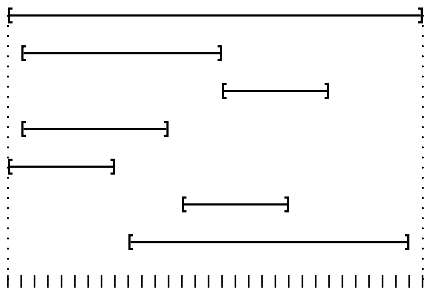
Optimized parameters for the upper case characters:

Error:
0.0473317

Directional Mapping:



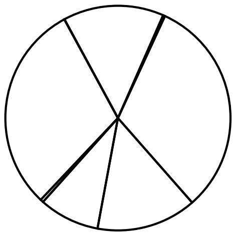
Activity Regions:	Scalar Bias:
[0,31]	177.192
[1,16]	143.307
[16,24]	107.94
[1,12]	104.505
[0,8]	141.477
[13,21]	126.338
[9,30]	139.984



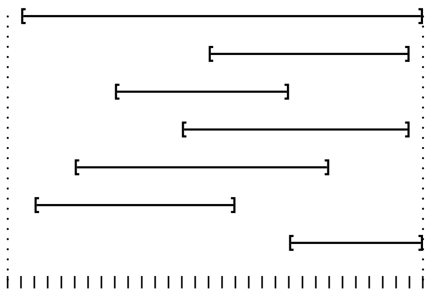
Optimized parameters for the lower case characters:

Error:
0.0491867

Directional Mapping:



Activity Regions:	Scalar Bias:
[1,31]	196.957
[15,30]	63.1508
[8,21]	181.011
[13,30]	153.856
[5,24]	69.2795
[2,17]	123.446
[21,31]	102.907

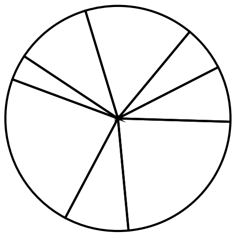


Subject “c38”

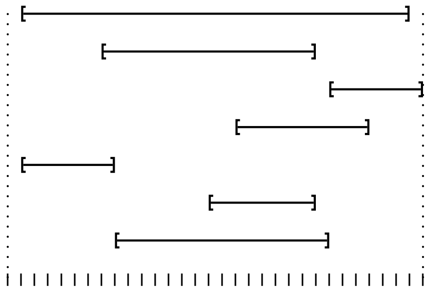
Optimized parameters for the upper case characters:

Error:
0.0775641

Directional Mapping:



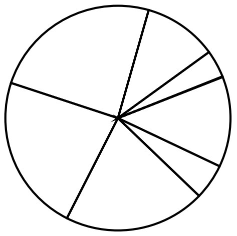
Activity Regions:	Scalar Bias:
[1,30]	110.493
[7,23]	102.245
[24,31]	188.424
[17,27]	99.3315
[1,8]	19.3787
[15,23]	78.63
[8,24]	80.6399



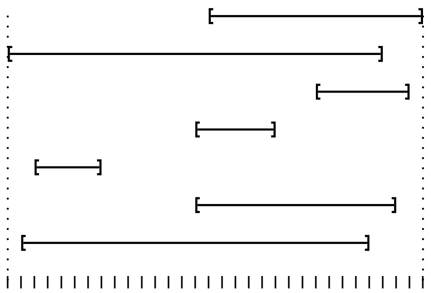
Optimized parameters for the lower case characters:

Error:
0.125401

Directional Mapping:



Activity Regions:	Scalar Bias:
[15,31]	70.9789
[0,28]	158.347
[23,30]	101.396
[14,20]	2.62904
[2,7]	130.156
[14,29]	88.5502
[1,27]	2.70559

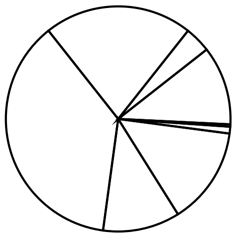


Subject “c39”

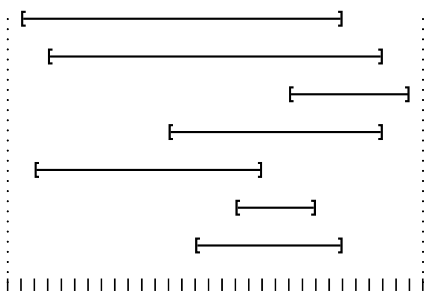
Optimized parameters for the upper case characters:

Error:
0.100505

Directional Mapping:



Activity Regions:

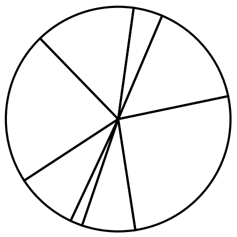


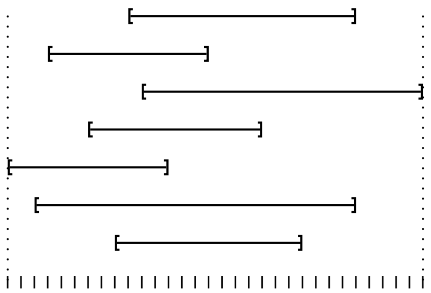
Activity Regions:	Scalar Bias:
[1,25]	170.992
[3,28]	123.701
[21,30]	195.373
[12,28]	174.717
[2,19]	183.57
[17,23]	80.6935
[14,25]	96.0666

Optimized parameters for the lower case characters:

Error:
0.0586739

Directional Mapping:





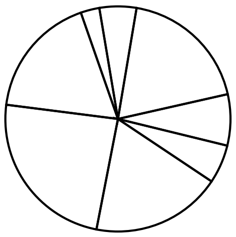
Activity Regions:	Scalar Bias:
[9,26]	12.7671
[3,15]	10.0789
[10,31]	71.8657
[6,19]	50.8172
[0,12]	75.6428
[2,26]	53.7131
[8,22]	120.695

Subject “c40”

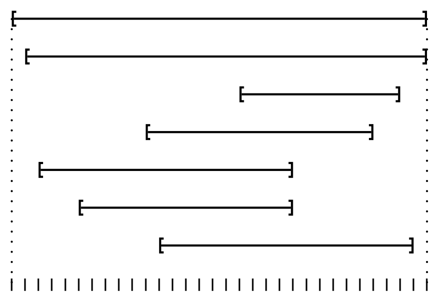
Optimized parameters for the upper case characters:

Error:
0.0850962

Directional Mapping:



Activity Regions:

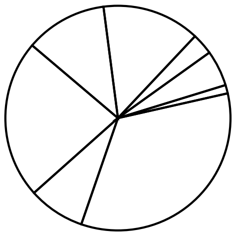


Activity Regions:	Scalar Bias:
[0,31]	148.319
[1,31]	159.939
[17,29]	44.3837
[10,27]	142.52
[2,21]	139.881
[5,21]	63.2344
[11,30]	30.3998

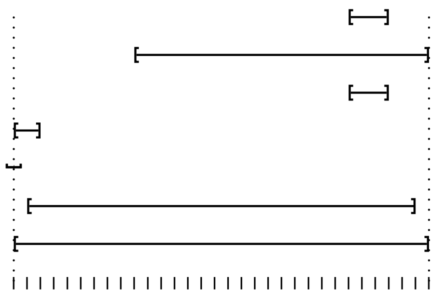
Optimized parameters for the lower case characters:

Error:
0.108562

Directional Mapping:



Activity Regions:



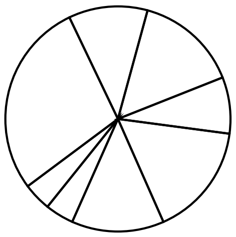
Activity Regions:	Scalar Bias:
[25,28]	68.5755
[9,31]	201.739
[25,28]	6.01585
[0,2]	191.438
[0,0]	129.89
[1,30]	83.319
[0,31]	81.7186

Subject “c41”

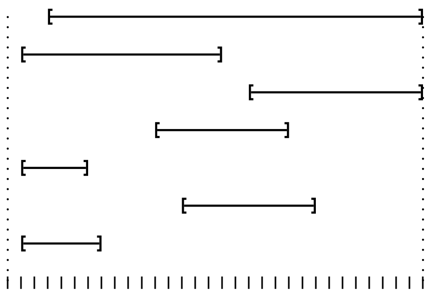
Optimized parameters for the upper case characters:

Error:
0.132216

Directional Mapping:



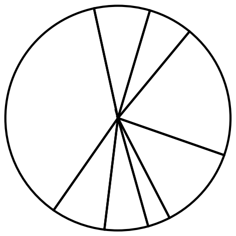
Activity Regions:	Scalar Bias:
[3,31]	98.3765
[1,16]	121.421
[18,31]	123.306
[11,21]	8.77814
[1,6]	16.6685
[13,23]	74.8922
[1,7]	100.982



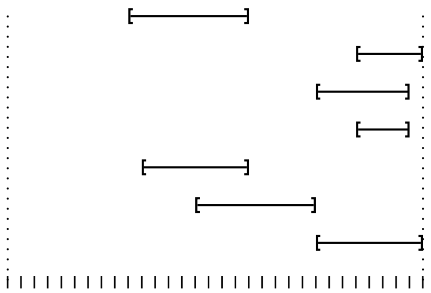
Optimized parameters for the lower case characters:

Error:
0.246314

Directional Mapping:



Activity Regions:	Scalar Bias:
[9,18]	0.00119558
[26,31]	128.49
[23,30]	99.1444
[26,30]	147.522
[10,18]	21.076
[14,23]	127.078
[23,31]	22.3664

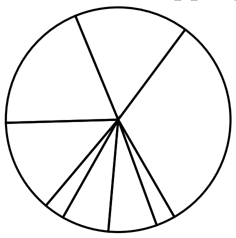


Subject "c42"

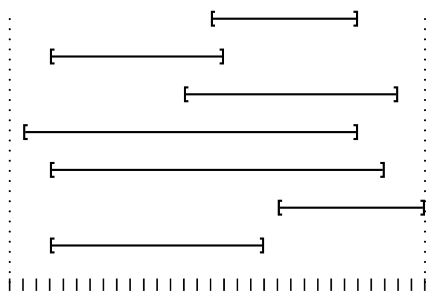
Optimized parameters for the upper case characters:

Error:
0.122889

Directional Mapping:



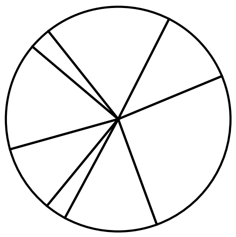
Activity Regions:	Scalar Bias:
[15,26]	78.7633
[3,16]	105.665
[13,29]	65.253
[1,26]	20.165
[3,28]	143.221
[20,31]	62.2536
[3,19]	8.79207



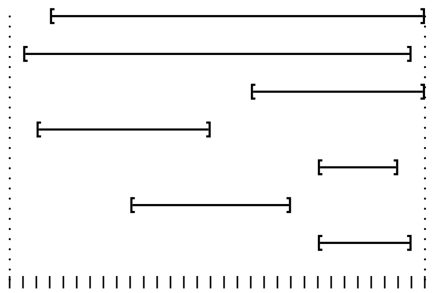
Optimized parameters for the lower case characters:

Error:
0.112196

Directional Mapping:



Activity Regions:	Scalar Bias:
[3,31]	137.532
[1,30]	16.4693
[18,31]	118.58
[2,15]	117.008
[23,29]	14.4251
[9,21]	146.075
[23,30]	51.9192

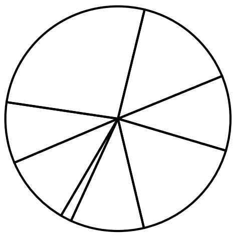


Subject “c43”

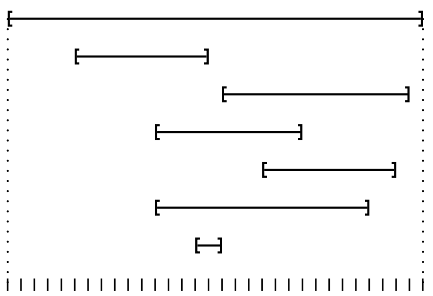
Optimized parameters for the upper case characters:

Error:
0.0983974

Directional Mapping:



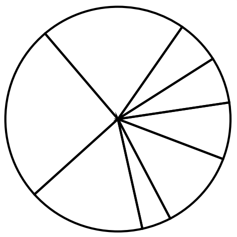
Activity Regions:	Scalar Bias:
[0,31]	178.697
[5,15]	94.0704
[16,30]	36.2074
[11,22]	67.4686
[19,29]	90.864
[11,27]	63.8612
[14,16]	11.2912



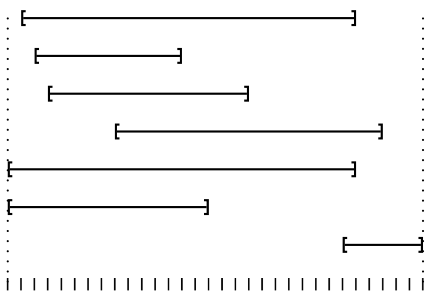
Optimized parameters for the lower case characters:

Error:
0.187364

Directional Mapping:



Activity Regions:	Scalar Bias:
[1,26]	32.2132
[2,13]	55.3336
[3,18]	37.821
[8,28]	76.658
[0,26]	88.7166
[0,15]	1.2372
[25,31]	71.2463

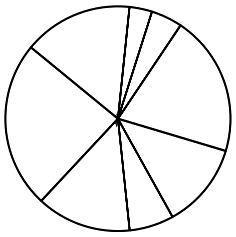


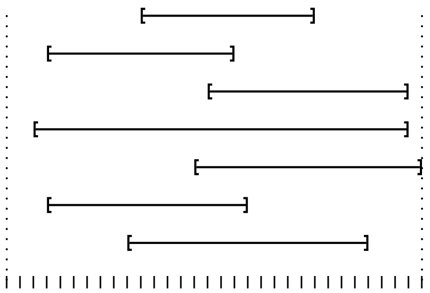
Subject “c44”

Optimized parameters for the upper case characters:

Error:
0.0816186

Directional Mapping:



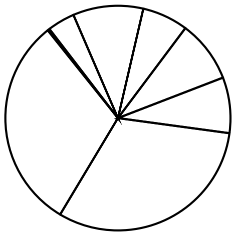


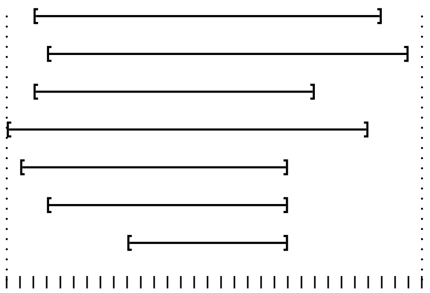
Activity Regions:	Scalar Bias:
[10,23]	62.8604
[3,17]	84.6672
[15,30]	10.7031
[2,30]	151.89
[14,31]	84.4914
[3,18]	0.593239
[9,27]	39.1273

Optimized parameters for the lower case characters:

Error:
0.0866747

Directional Mapping:





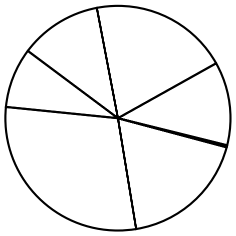
Activity Regions:	Scalar Bias:
[2,28]	27.1823
[3,30]	110.691
[2,23]	106.205
[0,27]	110.526
[1,21]	106.185
[3,21]	35.7056
[9,21]	101.35

Subject “t00”

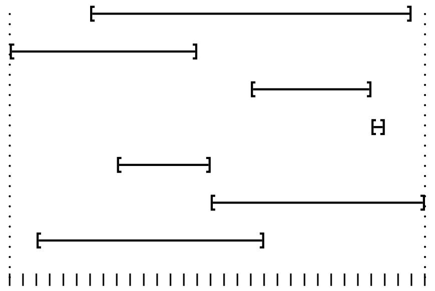
Optimized parameters for the upper case characters:

Error:
0.215917

Directional Mapping:



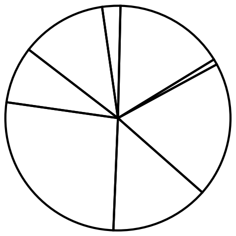
Activity Regions:	Scalar Bias:
[6,30]	149.113
[0,14]	4.44934
[18,27]	136.359
[27,28]	47.3665
[8,15]	89.2072
[15,31]	183.019
[2,19]	202.494



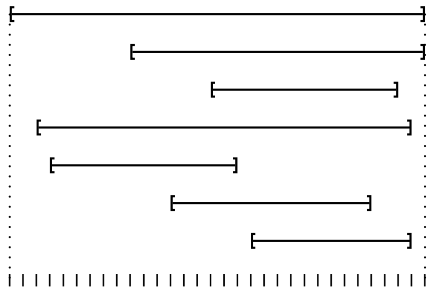
Optimized parameters for the lower case characters:

Error:
0.256639

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,31]	96.9642
[9,31]	41.1581
[15,29]	18.2992
[2,30]	75.2523
[3,17]	153.621
[12,27]	19.3605
[18,30]	185.628

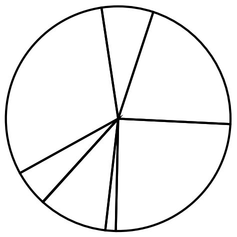


Subject “t01”

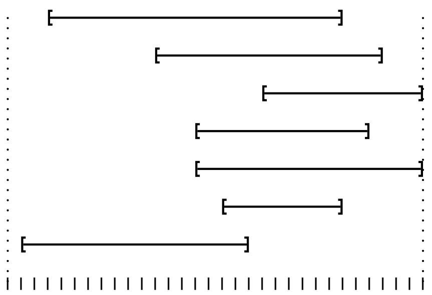
Optimized parameters for the upper case characters:

Error:
0.133085

Directional Mapping:



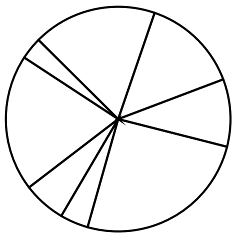
Activity Regions:	Scalar Bias:
[3,25]	71.0806
[11,28]	31.9444
[19,31]	97.0806
[14,27]	23.3147
[14,31]	106.558
[16,25]	163.182
[1,18]	80.5865



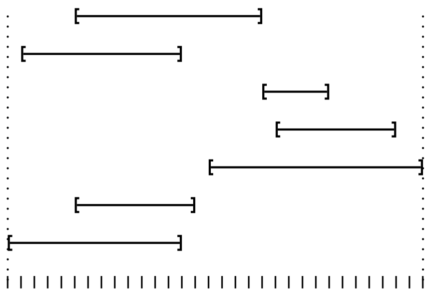
Optimized parameters for the lower case characters:

Error:
0.146811

Directional Mapping:



Activity Regions:	Scalar Bias:
[5,19]	49.0315
[1,13]	27.3605
[19,24]	6.19609
[20,29]	116.796
[15,31]	88.2369
[5,14]	68.031
[0,13]	64.4197

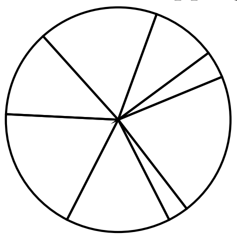


Subject “t02”

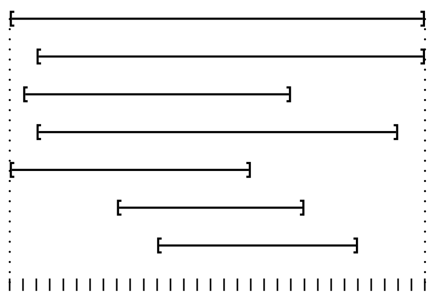
Optimized parameters for the upper case characters:

Error:
0.0786298

Directional Mapping:



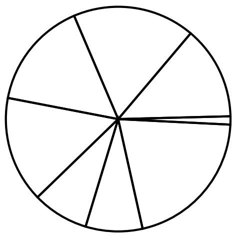
Activity Regions:	Scalar Bias:
[0,31]	76.7353
[2,31]	86.9643
[1,21]	63.8668
[2,29]	27.019
[0,18]	96.3404
[8,22]	121.287
[11,26]	32.2721



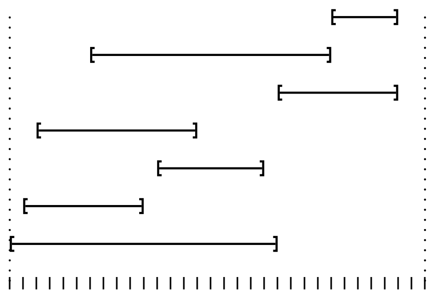
Optimized parameters for the lower case characters:

Error:
0.15397

Directional Mapping:



Activity Regions:	Scalar Bias:
[24,29]	29.2716
[6,24]	124.25
[20,29]	133.176
[2,14]	5.96883
[11,19]	104.623
[1,10]	61.9133
[0,20]	64.441

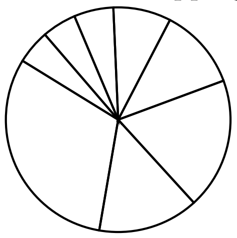


Subject “t03”

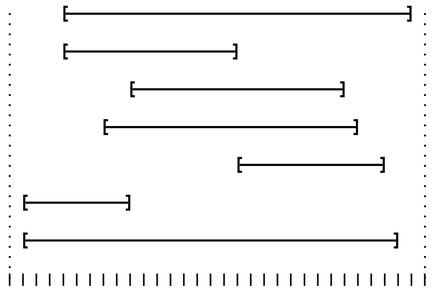
Optimized parameters for the upper case characters:

Error:
0.130172

Directional Mapping:



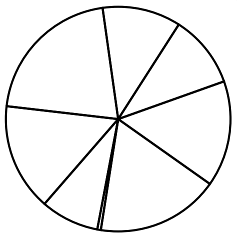
Activity Regions:	Scalar Bias:
[4,30]	112.066
[4,17]	115.404
[9,25]	92.3764
[7,26]	70.3925
[17,28]	39.7668
[1,9]	124.23
[1,29]	175.628



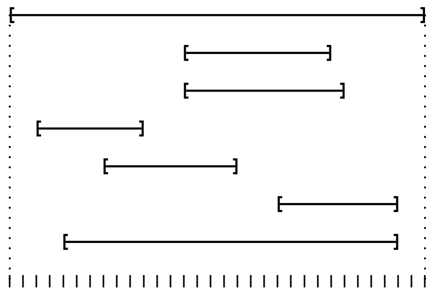
Optimized parameters for the lower case characters:

Error:
0.184784

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,31]	121.01
[13,24]	14.3915
[13,25]	41.6941
[2,10]	98.9317
[7,17]	149.451
[20,29]	168.482
[4,29]	126.465

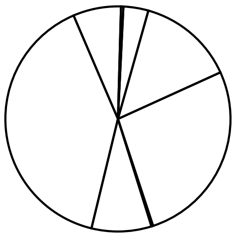


Subject “t04”

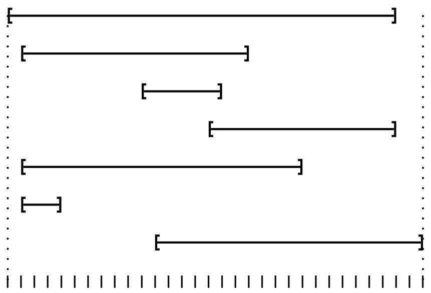
Optimized parameters for the upper case characters:

Error:
0.0763301

Directional Mapping:



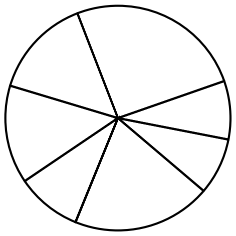
Activity Regions:	Scalar Bias:
[0,29]	179.047
[1,18]	153.068
[10,16]	12.8584
[15,29]	157.244
[1,22]	80.7516
[1,4]	15.3259
[11,31]	157.249



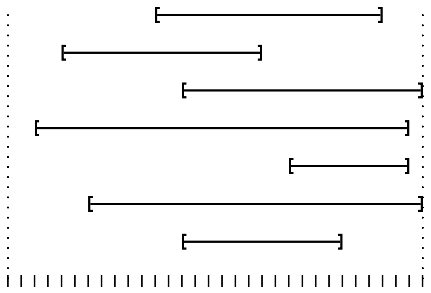
Optimized parameters for the lower case characters:

Error:
0.105773

Directional Mapping:



Activity Regions:	Scalar Bias:
[11,28]	110.455
[4,19]	186.579
[13,31]	190.65
[2,30]	136.434
[21,30]	185.344
[6,31]	59.3605
[13,25]	35.3756

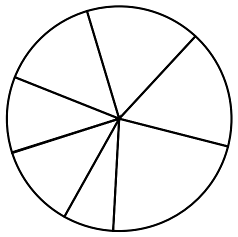


Subject "t05"

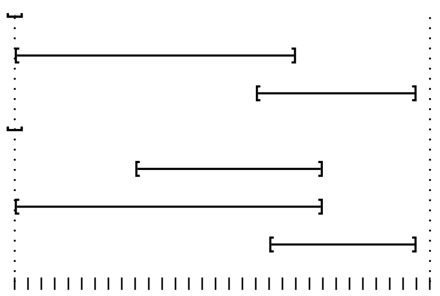
Optimized parameters for the upper case characters:

Error:
0.0539503

Directional Mapping:



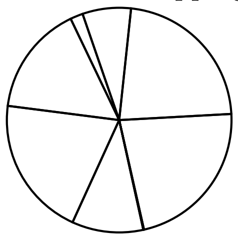
Activity Regions:	Scalar Bias:
[0,0]	168.149
[0,21]	147.414
[18,30]	176.478
[0,0]	161.472
[9,23]	149.933
[0,23]	66.1713
[19,30]	21.7509



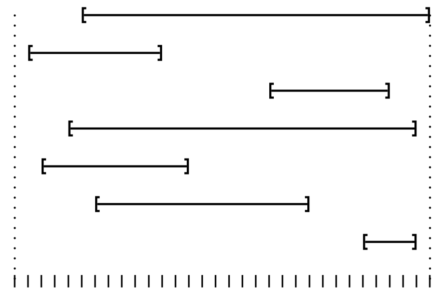
Optimized parameters for the lower case characters:

Error:
0.0834455

Directional Mapping:



Activity Regions:	Scalar Bias:
[5,31]	121.864
[1,11]	88.2476
[19,28]	147.191
[4,30]	6.53906
[2,13]	94.6447
[6,22]	152.666
[26,30]	10.3714

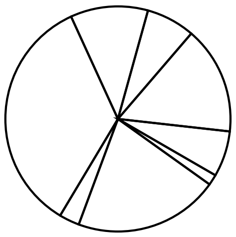


Subject "t06"

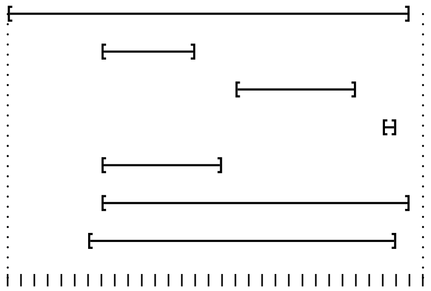
Optimized parameters for the upper case characters:

Error:
0.0832011

Directional Mapping:



Activity Regions:

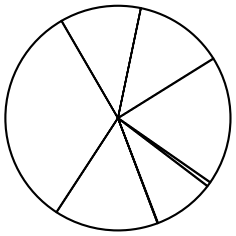


Activity Regions:	Scalar Bias:
[0,30]	186.804
[7,14]	43.1142
[17,26]	117.448
[28,29]	46.9402
[7,16]	158.354
[7,30]	23.6205
[6,29]	119.947

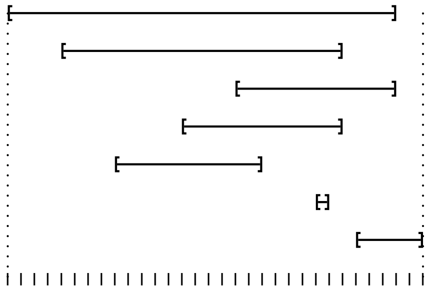
Optimized parameters for the lower case characters:

Error:
0.143854

Directional Mapping:



Activity Regions:



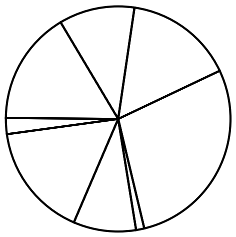
Activity Regions:	Scalar Bias:
[0,29]	152.859
[4,25]	134.167
[17,29]	172.901
[13,25]	91.0565
[8,19]	101.41
[23,24]	43.8918
[26,31]	83.154

Subject “t07”

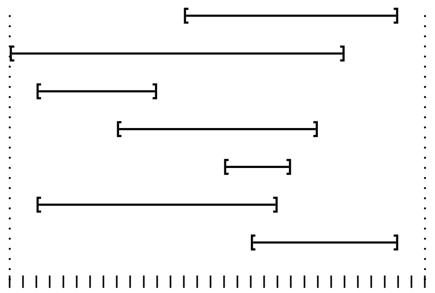
Optimized parameters for the upper case characters:

Error:
0.0553325

Directional Mapping:



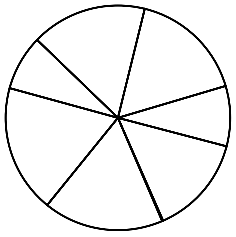
Activity Regions:	Scalar Bias:
[13,29]	26.7755
[0,25]	101.976
[2,11]	20.3077
[8,23]	129.457
[16,21]	13.589
[2,20]	119.999
[18,29]	186.066



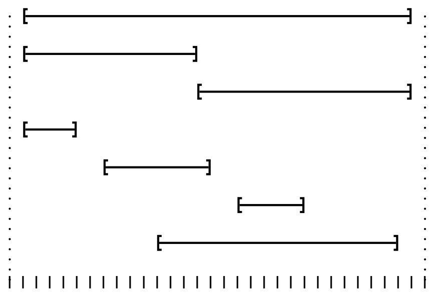
Optimized parameters for the lower case characters:

Error:
0.0509014

Directional Mapping:



Activity Regions:	Scalar Bias:
[1,30]	155.29
[1,14]	101.538
[14,30]	86.5866
[1,5]	98.3551
[7,15]	94.6962
[17,22]	58.3699
[11,29]	102.452

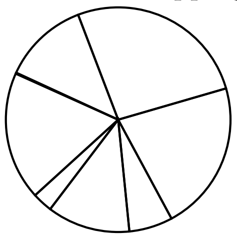


Subject “t08”

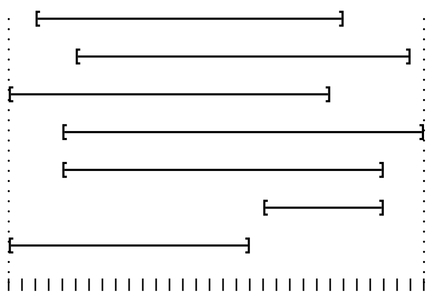
Optimized parameters for the upper case characters:

Error:
0.099403

Directional Mapping:



Activity Regions:

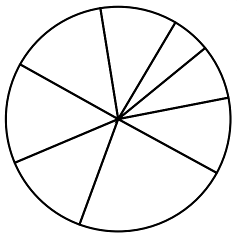


Activity Regions:	Scalar Bias:
[2,25]	99.8
[5,30]	74.2093
[0,24]	122.851
[4,31]	127.454
[4,28]	70.4073
[19,28]	97.1993
[0,18]	151.153

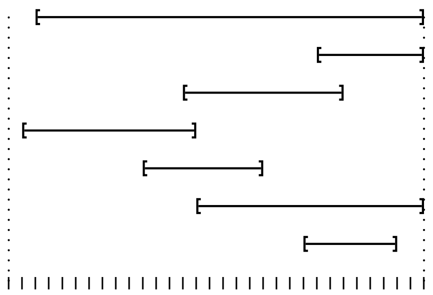
Optimized parameters for the lower case characters:

Error:
0.0911298

Directional Mapping:



Activity Regions:



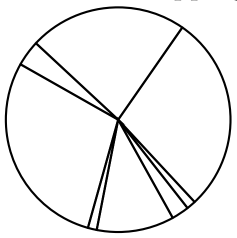
Activity Regions:	Scalar Bias:
[2,31]	148.964
[23,31]	10.1868
[13,25]	113.072
[1,14]	77.2711
[10,19]	109.64
[14,31]	3.94666
[22,29]	95.3894

Subject “t09”

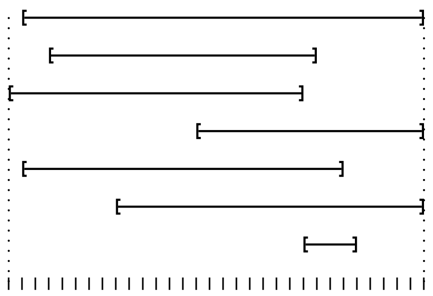
Optimized parameters for the upper case characters:

Error:
0.181386

Directional Mapping:



Activity Regions:

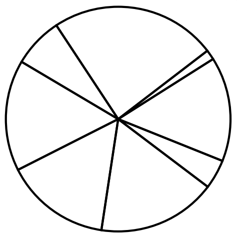


Activity Regions:	Scalar Bias:
[1,31]	134.765
[3,23]	110.666
[0,22]	112.737
[14,31]	194.949
[1,25]	98.2005
[8,31]	138.815
[22,26]	3.13344

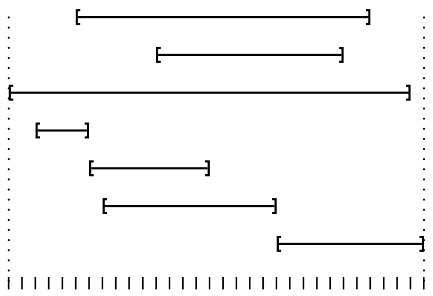
Optimized parameters for the lower case characters:

Error:
0.108886

Directional Mapping:



Activity Regions:



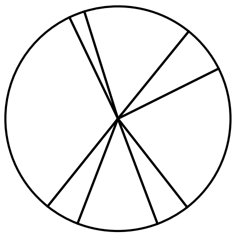
Activity Regions:	Scalar Bias:
[5,27]	24.4772
[11,25]	66.9013
[0,30]	169.179
[2,6]	18.6468
[6,15]	183.064
[7,20]	82.8069
[20,31]	167.304

Subject “t10”

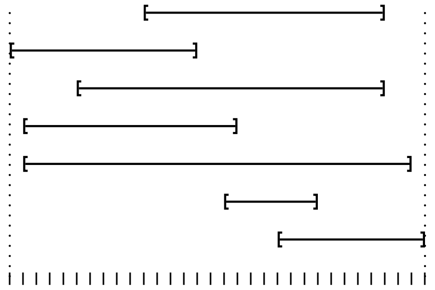
Optimized parameters for the upper case characters:

Error:
0.111687

Directional Mapping:



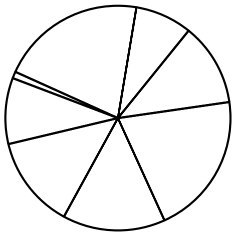
Activity Regions:	Scalar Bias:
[10,28]	168.515
[0,14]	81.6709
[5,28]	11.2988
[1,17]	96.281
[1,30]	124.975
[16,23]	65.3763
[20,31]	111.276



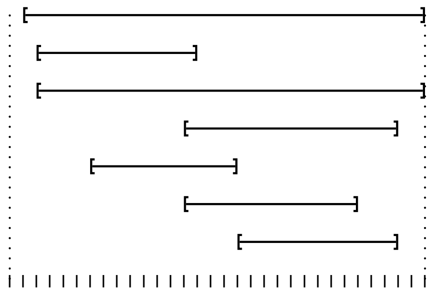
Optimized parameters for the lower case characters:

Error:
0.167732

Directional Mapping:



Activity Regions:	Scalar Bias:
[1,31]	161.712
[2,14]	183.021
[2,31]	26.0808
[13,29]	30.6264
[6,17]	45.5874
[13,26]	106.871
[17,29]	159.703

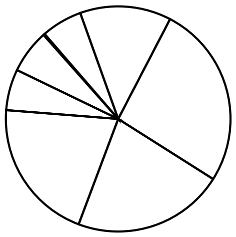


Subject “t11”

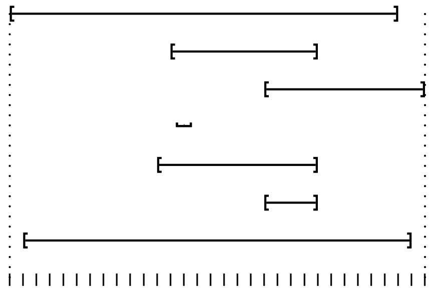
Optimized parameters for the upper case characters:

Error:
0.177007

Directional Mapping:



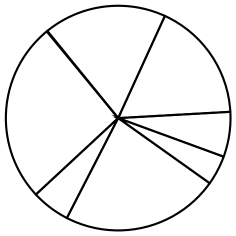
Activity Regions:	Scalar Bias:
[0,29]	79.2241
[12,23]	0.39717
[19,31]	159.404
[13,13]	192.048
[11,23]	139.694
[19,23]	32.5767
[1,30]	71.0145



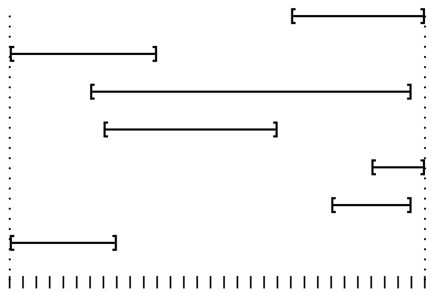
Optimized parameters for the lower case characters:

Error:
0.182107

Directional Mapping:



Activity Regions:	Scalar Bias:
[21,31]	72.1158
[0,11]	31.8859
[6,30]	177.329
[7,20]	73.0161
[27,31]	133.366
[24,30]	185.054
[0,8]	81.933

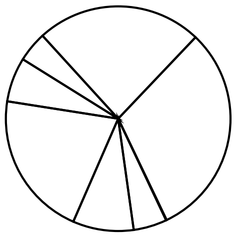


Subject “t12”

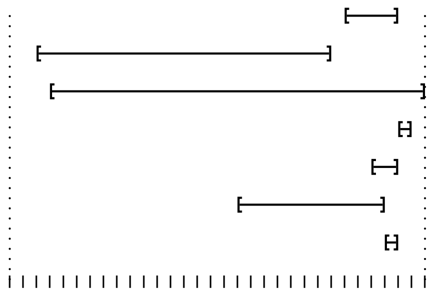
Optimized parameters for the upper case characters:

Error:
0.104159

Directional Mapping:



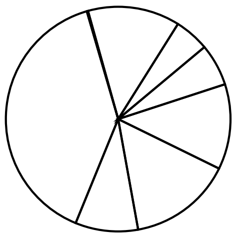
Activity Regions:	Scalar Bias:
[25,29]	36.4633
[2,24]	137.513
[3,31]	156.017
[29,30]	44.1485
[27,29]	30.7142
[17,28]	146.33
[28,29]	122.316



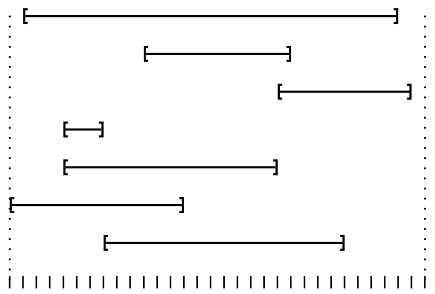
Optimized parameters for the lower case characters:

Error:
0.100016

Directional Mapping:



Activity Regions:	Scalar Bias:
[1,29]	188.353
[10,21]	0.718574
[20,30]	194.716
[4,7]	12.5296
[4,20]	179.027
[0,13]	114.107
[7,25]	31.1494

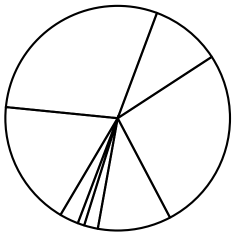


Subject “t13”

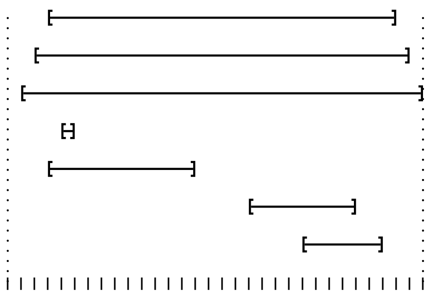
Optimized parameters for the upper case characters:

Error:
0.0819111

Directional Mapping:



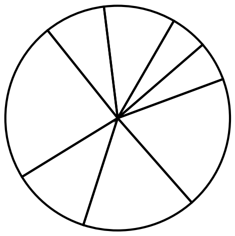
Activity Regions:	Scalar Bias:
[3,29]	98.314
[2,30]	129.353
[1,31]	4.87348
[4,5]	29.4743
[3,14]	191.215
[18,26]	167.264
[22,28]	177.552



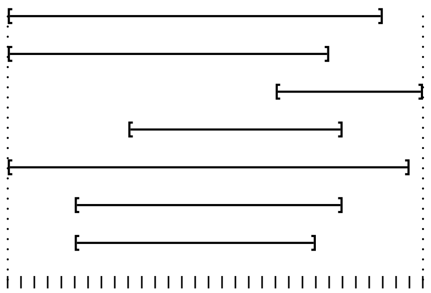
Optimized parameters for the lower case characters:

Error:
0.0641627

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,28]	98.3724
[0,24]	26.189
[20,31]	135.425
[9,25]	21.7641
[0,30]	40.5988
[5,25]	38.3009
[5,23]	24.6958

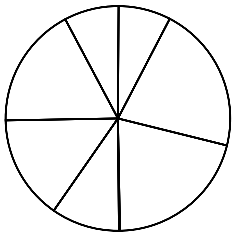


Subject “t14”

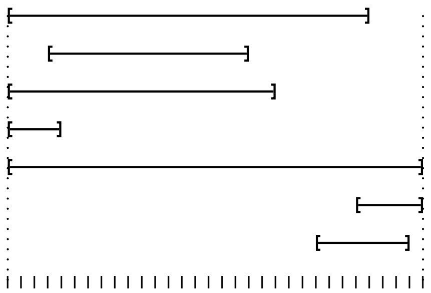
Optimized parameters for the upper case characters:

Error:
0.103329

Directional Mapping:



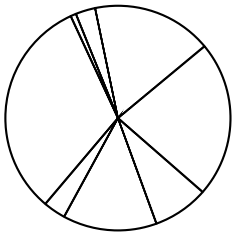
Activity Regions:	Scalar Bias:
[0,27]	96.7879
[3,18]	176.406
[0,20]	78.4025
[0,4]	126.488
[0,31]	18.58
[26,31]	33.7079
[23,30]	189.252



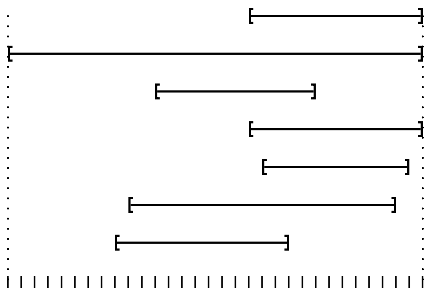
Optimized parameters for the lower case characters:

Error:
0.0946955

Directional Mapping:



Activity Regions:	Scalar Bias:
[18,31]	140.54
[0,31]	165.358
[11,23]	62.3377
[18,31]	107.704
[19,30]	54.6595
[9,29]	88.9864
[8,21]	81.6355

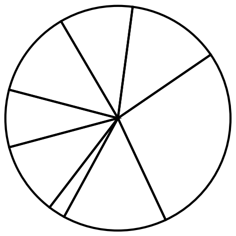


Subject "t15"

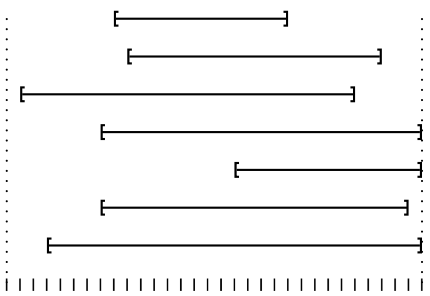
Optimized parameters for the upper case characters:

Error:
0.186298

Directional Mapping:



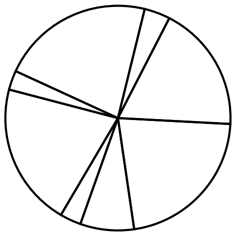
Activity Regions:	Scalar Bias:
[8,21]	145.163
[9,28]	10.8799
[1,26]	102.439
[7,31]	71.11
[17,31]	134.845
[7,30]	20.9933
[3,31]	71.0725



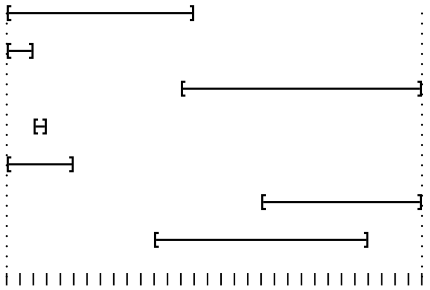
Optimized parameters for the lower case characters:

Error:
0.195068

Directional Mapping:



Activity Regions:	Scalar Bias:
[0,14]	95.9962
[0,2]	183.977
[13,31]	94.298
[2,3]	46.6957
[0,5]	143.999
[19,31]	122.289
[11,27]	87.3799

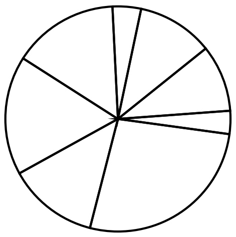


Subject “t16”

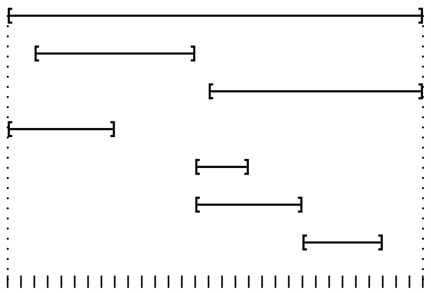
Optimized parameters for the upper case characters:

Error:
0.0477163

Directional Mapping:



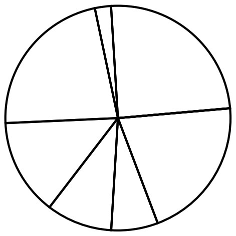
Activity Regions:	Scalar Bias:
[0,31]	168.476
[2,14]	129.387
[15,31]	144.956
[0,8]	198.688
[14,18]	36.1123
[14,22]	147.164
[22,28]	67.5201



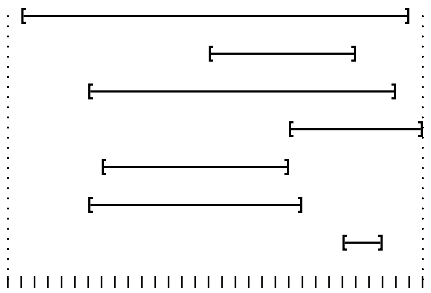
Optimized parameters for the lower case characters:

Error:
0.0302965

Directional Mapping:



Activity Regions:	Scalar Bias:
[1,30]	85.6707
[15,26]	114.221
[6,29]	48.6986
[21,31]	134.688
[7,21]	160.101
[6,22]	31.2551
[25,28]	103.128

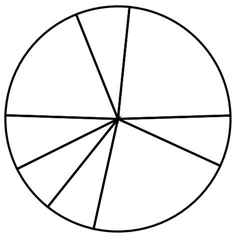


Subject "t17"

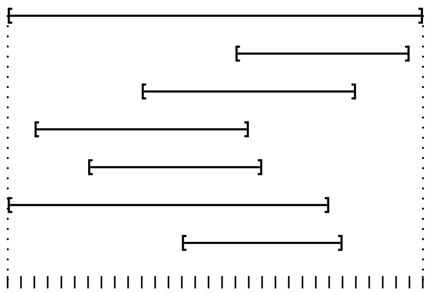
Optimized parameters for the upper case characters:

Error:
0.15484

Directional Mapping:



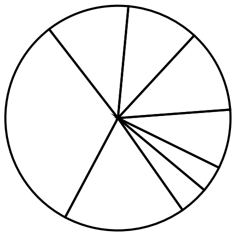
Activity Regions:	Scalar Bias:
[0,31]	64.2141
[17,30]	87.663
[10,26]	50.9263
[2,18]	117.559
[6,19]	7.47494
[0,24]	32.5805
[13,25]	85.4686



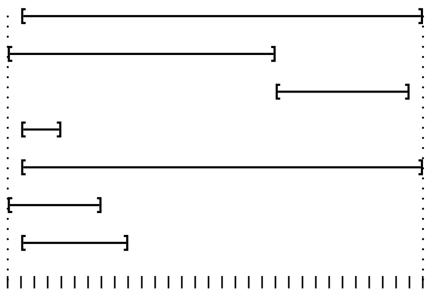
Optimized parameters for the lower case characters:

Error:
0.1226

Directional Mapping:



Activity Regions:	Scalar Bias:
[1,31]	90.0627
[0,20]	176.759
[20,30]	123.344
[1,4]	170.476
[1,31]	2.8479
[0,7]	137.33
[1,9]	117.779

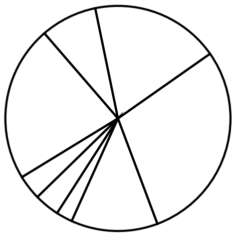


Subject "t18"

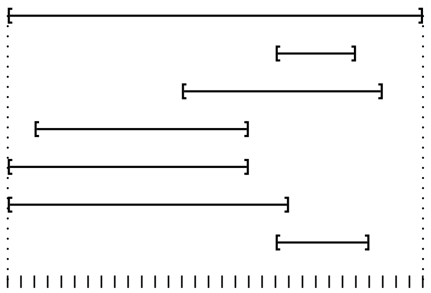
Optimized parameters for the upper case characters:

Error:
0.107131

Directional Mapping:



Activity Regions:

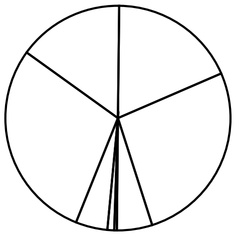


Activity Regions:	Scalar Bias:
[0,31]	122.974
[20,26]	61.6228
[13,28]	161.169
[2,18]	155.736
[0,18]	66.6944
[0,21]	61.3375
[20,27]	147.299

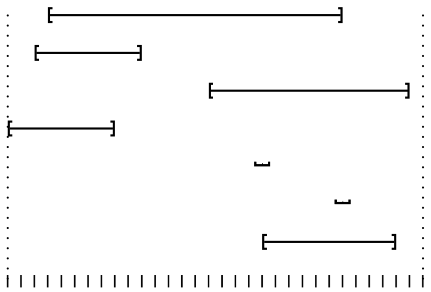
Optimized parameters for the lower case characters:

Error:
0.0971034

Directional Mapping:



Activity Regions:



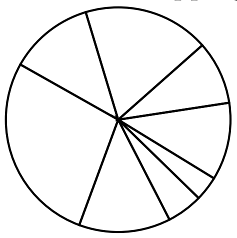
Activity Regions:	Scalar Bias:
[3,25]	123.881
[2,10]	5.05596
[15,30]	67.8729
[0,8]	157.443
[19,19]	186.331
[25,25]	141.508
[19,29]	134.309

Subject “t19”

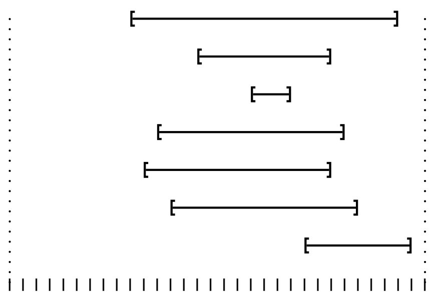
Optimized parameters for the upper case characters:

Error:
0.11655

Directional Mapping:



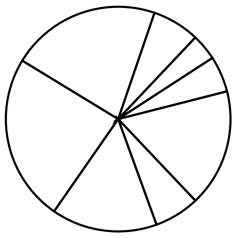
Activity Regions:	Scalar Bias:
[9,29]	36.8398
[14,24]	24.7286
[18,21]	18.7045
[11,25]	60.1853
[10,24]	65.9961
[12,26]	64.4141
[22,30]	136.062



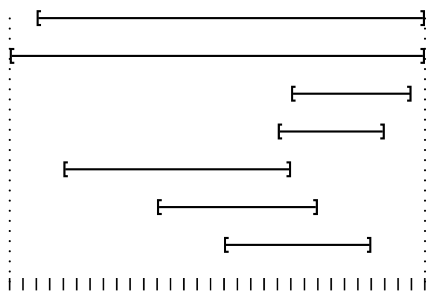
Optimized parameters for the lower case characters:

Error:
0.189291

Directional Mapping:



Activity Regions:	Scalar Bias:
[2,31]	32.0364
[0,31]	150.378
[21,30]	144.672
[20,28]	59.5064
[4,21]	182.121
[11,23]	85.4956
[16,27]	69.2216

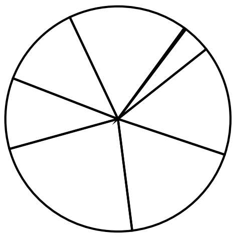


Subject “t20”

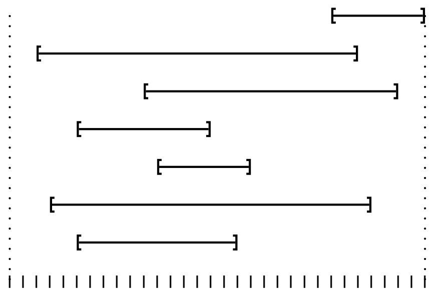
Optimized parameters for the upper case characters:

Error:
0.0841747

Directional Mapping:



Activity Regions:

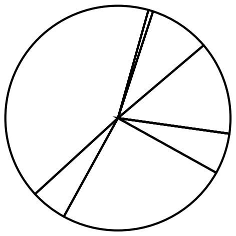


Activity Regions:	Scalar Bias:
[24,31]	192.768
[2,26]	142.46
[10,29]	181.425
[5,15]	33.6511
[11,18]	12.0134
[3,27]	134.11
[5,17]	177.909

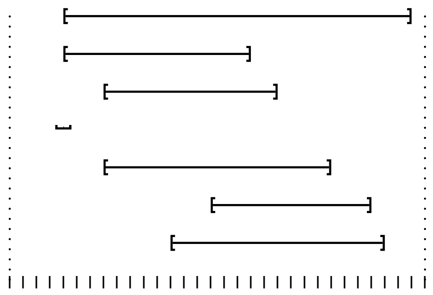
Optimized parameters for the lower case characters:

Error:
0.0772877

Directional Mapping:



Activity Regions:



Activity Regions:	Scalar Bias:
[4,30]	179.04
[4,18]	38.4046
[7,20]	94.4444
[4,4]	183.526
[7,24]	61.632
[15,27]	104.869
[12,28]	94.0085

Appendix C

CDROM Contents

Bound with this dissertation is a CDROM containing 739MB of data from the non-stylized English and optimization studies (Sections 6.2 and 6.3). The top level directories are each compressed with TAR and GZIP in order to meet the CDROM size limitation. Once each of these tarballs is decompressed you will find the directory structure shown in Figure C.1(A). This chapter reviews the contents of each directory, file naming conventions, and details the format of each file type.

C.1 Character Samples

The "`character_samples`" directory contains the character drawings collected in the non-stylized English study (Section 6.2) in their raw form. Each subject has a single data file named by their subject identifier plus "`.txt`". Thus, the example file name in Figure C.1(B) is for the samples drawn by subject "c00". These file names are the base file names used throughout the remaining directories on the CDROM.

Each file contains 1716 lines, one for each letter drawing provided by the subject. The lines have the following format:

$$C \quad P \quad S \quad x_0 \quad y_0 \ldots x_{P-1} \quad y_{P-1} \quad d_0 \quad u_0 \ldots d_{S-1} \quad u_{S-1}$$

C is the ASCII character represented by the line, P is the number of (X,Y) coordinate pairs in the drawing, and S is the number of strokes. Following this header information, each coordinate pair (x_i, y_i) is listed. Next the strokes are defined by pen down and up events.

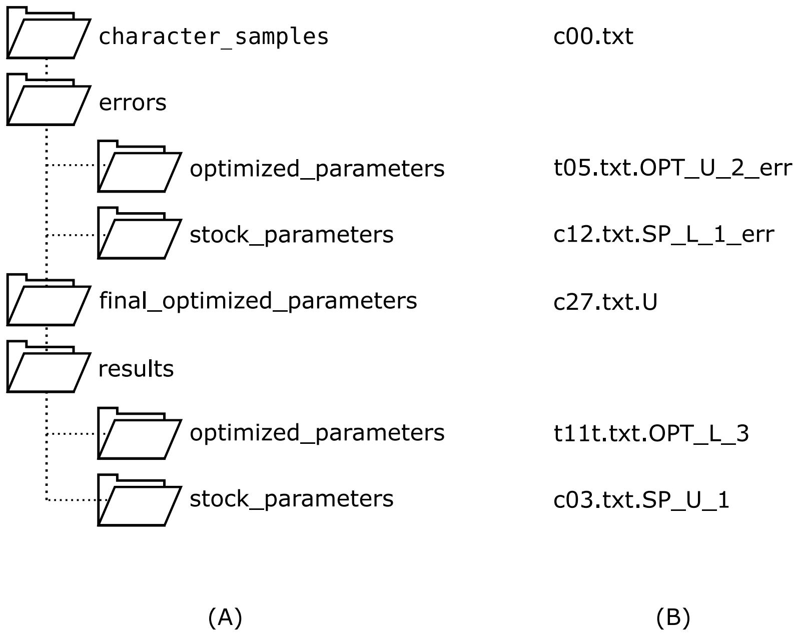


Figure C.1: (A) Directory structure on the CDROM and (B) example file names for each directory

d_i indicates which point in the drawing was the ith pen down event... specifically the ith pen down event occurs at the coordinate pair (x_{d_i}, y_{d_i}). Similarly, u_i indicates which point in the drawing was the ith pen up event.

C.2 Errors

The "`errors`" directory contains files listing the recognition errors resulting from the evaluation of a parmeter set for a specific letter-case and α value. These files are distributed into two directories, "`optimized_parameters`" and "`stock_parameters`", identifying whether the parameter set is the one optimized by the study in Section 6.3 or the stock set. The file extension begins with "`SP`" for stock parameters or "`OPT`" for an optimized set. The "`_U_`" and "`_L_`" extension flags indicate the letter-case, upper or lower respectively, and the numeral finishing the extension is the α value. The base name of each file identifies the subject. Thus, the example "`optimized_parameters`" file name in Figure C.1(B) is for the errors found recognizing the upper case samples drawn by subject "t05" using an optimized parameter set where $\alpha = 2$. The example "`stock_parameters`" file name in Figure C.1(B) is for the errors found recognizing the lower case samples drawn by subject "c12" using the stock parameter set where $\alpha = 1$.

Each file contains 900 lines, one for each random, α-sized alphabet tested. The lines have the following format:

$$N \quad c_0 \ldots c_{N-1} \quad t_0 \quad r_0 \quad t_1 \quad r_1 \ldots$$

$N = \alpha \times 26$ is the total number of character samples in the alphabet tested. c_i indicates which sample from the subject's sample file (Section C.1) the the ith member of the alphabet is. Specifically, line c_i (zero-based) is the ith sample for the current, random alphabet. Following the alphabet identification are value pairs indicating a single recognition error each (continuing to the end of the line). t_i is the drawing to be recognized and r_i is the drawing that was incorrectly determined to be the closest match. Similar to c_i, t_i and r_i are zero-based line numbers in the subject's sample file.

C.3 Final Optimization Parameters

The "`final_optimization_parameters`" directory contains the final, best fitness parameters found for each subject and letter case in the optimization study (Section 6.3). The base file name indicates the subject as in Section C.1, and the "`U`" and "`L`" extensions indicate whether the parameters apply to the upper or lower case characters respectively.

The example file name in Figure C.1(B) is for the the upper case parameters evolved for subject "c27". Additionally, each subject's parameters are visualized in Appendix B.

Each file contains a single line in the following format:

$$F \quad a_0 \ldots a_7 \quad s_0 \quad e_0 \ldots s_6 \quad e_6 \quad b_0 \ldots b_6$$

F is the fitness (error rate) measured over 300 random alphabets where $\alpha = 1$. a_i is the lower, non-inclusive bound of the ith angular region in the evolved directional mapping. s_i and e_i indicate the zero-based substrokes that start and end the ith activity region. b_i is the scalar bias applied to the ith activity region.

C.4 Results

The "`results`" directory contains the recognition results for subjects on each of the 900 random alphabets evaluated for upper and lower case letters. These files are distributed into two directories, "`optimized_parameters`" and "`stock_parameters`", identifying whether the parameter set is the one optimized by the study in Section 6.3 or the stock set. The file extension begins with "`SP`" for stock parameters or "`OPT`" for an optimized set. The "`_U_`" and "`_L_`" extension flags indicate the letter-case, upper or lower respectively, and the numeral finishing the extension is the α value. The base name of each file identifies the subject. The example "`optimized_parameters`" file name in Figure C.1(B) is for lower case samples drawn by subject "t11" using an optimized parameter set where $\alpha = 3$. The example "`stock_parameters`" file name in Figure C.1(B) is for upper case samples drawn by subject "c03" using the stock parameter set where $\alpha = 1$.

Each file contains 900 lines, one for each random alphabet. Lines are formatted as follows:

$$T \quad e_1 \ldots e_26$$

$T = \sum_{i=1}^{26} e_i$ where e_i is the number of ith letter's drawings that were misrecognized on the run. In this notation i refers to a letter of the alphabet where 'a'$= 1$ and 'z'$= 26$ (in the lower case, for example).