Assist Channel Coding for Improving Optical Character Recognition

by

Eric Heng-Shiang Kuo

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2000

© Eric Heng-Shiang Kuo, MM. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author

Department of Electrical Engineering and Computer Science

Certified by

Tommi S. Jaakkola
Assistant Professor
Thesis Supervisor

Accepted by

Arthur C. Smith
Chairman, Department Committee on Graduate Students
Assist Channel Coding for Improving Optical Character Recognition

by

Eric Heng-Shiang Kuo

Submitted to the Department of Electrical Engineering and Computer Science on May 22, 2000, in partial fulfillment of the requirements for the degrees of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science

Abstract

An assist channel is a compact, efficient, machine-readable encoding of information to improve the accuracy of the optical character recognition (OCR) processing of a document. Two primary techniques are developed for designing an assist channel. The technique of separation coding is a version of data compression developed to eliminate the most common errors made by an OCR engine. Error correcting codes are used to detect all errors that occur on a page of text. Experiments with physical documents are done to determine the characteristics of a specific OCR engine, and to determine the optimal design for an assist channel.

Thesis Supervisor: Tommi S. Jaakkola
Title: Assistant Professor
Acknowledgments

In addition to my thesis supervisor, Tommi Jaakkola, I especially would like to thank the people at Xerox PARC who made this project possible. Great thanks go to Daniel Greene as my immediate supervisor at PARC for coming up with the idea and giving me guidance. Thanks also go out to Marshall Bern, Kris Popat, and Lorna Fear for their valuable support. And finally, many thanks to my parents for keeping track of my progress.
Contents

1 Introduction 7
  1.1 Background ............................................. 7
  1.2 Techniques ............................................ 8
  1.3 Organization of Thesis ................................. 9

2 Optical Character Recognition models 11
  2.1 Pattern-matching model .............................. 11
  2.2 Document Image Decoding ............................ 13
  2.3 Other OCR models .................................... 14

3 Separation Coding 15
  3.1 Purpose and Description ............................. 15
  3.2 Design .................................................. 16
  3.3 Building Algorithm for the Separation Code ....... 17

4 Error Correcting Codes 19
  4.1 Introduction ........................................... 19
  4.2 Use of Error Correction in OCR .................... 20
  4.3 Models for Error Simulation ......................... 22
     4.3.1 One desynchronization error per row ............. 22
     4.3.2 Runs of Errors .................................. 23
  4.4 Computing Statistical Functions ..................... 27
  4.5 Comparing Block Sizes ................................ 28
# List of Tables

4.1 The probabilities of runs of length $i$ occurring (original documents) 30
4.2 Error statistics for different segment lengths (original documents) 31
4.3 Block failure rates for original set of documents 32
4.4 Page failure rates for original set of documents 32
4.5 Required error-correction strength for original documents 32

5.1 Required error-correction strength for fifth-hand copies of original documents 37
5.2 Required error-correction strength for fifth-hand copies of modified documents 38

A.1 Probability distribution of runs of length $i$ in fifth-hand copies of original documents 46
A.2 Probability distribution of runs of length $i$ in fifth-hand copies of modified documents 46

B.1 Error statistics for different segment lengths (fifth-hand copies of original documents) 47
B.2 Block failure rates for original set of documents 48
B.3 Page failure rates for original set of documents 48
B.4 Error statistics for different segment lengths (fifth-hand copies of edited documents) 48
B.5 Block failure rates for fifth-hand copies of edited documents 49
B.6 Page failure rates for fifth-hand copies of modified documents 49
Chapter 1

Introduction

Imagine that two versions of a particular document are available, one physically, the other electronically. Suddenly, a virus eats up the electronic version. The only way to restore it would be to retype the entire document. But retyping the document would be a tedious, time-consuming task. So instead, scanning is tried, and the text is read in by optical character recognition (OCR). But the OCR may prove to be error-prone. What can be done to reduce the error rate? When processing the document, include an assist channel!

An assist channel is a compact, machine-readable encoding of information to improve the optical character recognition (OCR) processing of a document. This channel may be situated in one of the margins, or perhaps spread out in small blocks after each paragraph. The channel may consist of machine-readable glyph codes that can easily be printed on the document and read by a machine. The desired purpose of an assist channel is to minimize the errors in reading the main document. In addition, assist channels should carry just enough information so that it takes up as little space as possible.

1.1 Background

Assist channel coding is the technology that bridges the physical paper world with the digital world. It is useful in cases where the online availability of the document is
uncertain. The electronic version of a document may have been corrupted, deleted, or inaccessible. The assist channel would help provide an accurate reconstruction of the electronic version from the physical version.

There are three important properties that assist channels should have. First, they need to do its job and reduce the error rate in the OCR process. Second, they should leave the primary text unchanged and use up as little space as possible. Third, the algorithms that surround assist channels should be fast and simple. Finding a design that accommodates all three properties is a challenge. The assist channel might guarantee a 100% reliability rate, but it might take up lots of space and work very slowly. Otherwise, it may occupy little space and be processed very fast, but the reliability of the OCR may be hardly improved. There is a large design space to work with, and the goal is to come as close to the center of this design space as possible.

Lopresti and Sandberg have done previous work to improve OCR by placing a bar code in the side channel [4]. On each page, the bar code contains a certificate that is computed for that page. In their scheme, for each line there are two entries stored in the bar code: the decimation and the hash. The decimation consists of one bit per character, and the hash is a 32-bit value computed over the entire line. This is to ensure that no characters were left out or split into two characters.

Other work has been done to design fonts to make them OCR-friendly. However, modifying fonts implies a change to the primary text. The goal of assist channel coding is to leave the appearance of the text unchanged.

1.2 Techniques

Two techniques that could be useful for assist channel coding are separation coding and error correction. These techniques or some combination of them will enable the assist channel to improve the accuracy of the OCR.

Separation coding is designed to prevent the most common mistakes in OCR. For instance, the characters l (lowercase L) and 1 (numeral one) look very similar, as do the period and comma. Thus the idea of separation coding is born: partition the
set of characters into groups so that similar characters get placed in different groups. Each group is assigned a label. The text would then be encoded by replacing each character with the label of the group that contains that character. The encoding becomes the assist channel.

In the error-correction process, the assist channel contains the check digits that should match up with the digits of the primary text. Typically, in an error-correcting code, the text is divided vertically into blocks. Each block could consist of the nth character in each row, and the check digits would be in the assist channel at the bottom of the page. When an error is detected, the line in which the error occurred will need to be reinterpreted. For instance, if $m$ had been mistakenly interpreted as $rn$, each block would have an error on that line since the correct character would be to the right. After the reinterpretation, the next block can be checked.

It is possible to combine these techniques so that the advantages of each technique are incorporated into the assist channel. In this case, the set of characters is partitioned into labeled groups as before, and the page is divided vertically into blocks. The digits of the block would be the group labels of the characters in the block. Instead of placing a direct encoding into the assist channel, the error correction check digits occupy the assist channel. In addition, a second error correction procedure would be done to correct any substitutions of characters within the same group. A second set of check digits is put into the assist channel for that purpose.

One additional technique that may be useful is document image decoding (DID) [2]. This technique is based on Markov models used in speech recognition. It is possible that the techniques of assist channel coding can be coupled with DID.

1.3 Organization of Thesis

Chapter 2 discusses the different models of optical character recognition. Assumptions about the pattern-matching model are made, and the confusion matrix is introduced.

In Chapter 3, separation coding is explained in more detail. Algorithms are designed to find the best separation code so that the error rate is minimized, given that
the maximum entropy is $n$ bits.

Chapter 4 describes the error-correction process with assist channels. A combinatorial theory is developed to model the generation of errors in the OCR process. Statistical techniques are developed to compute the optimal block size for a page based on the error rates of the OCR process.

Chapter 5 explores the accuracy of the modeling of the OCR process. Methods of gathering confusion data are explained. Once the data is gathered, computations are made according to the theories developed in the previous chapters. Some interesting observations are discussed.

Chapter 6 shows how the techniques of separation coding and error correction can be combined to improve OCR. And finally, Chapter 7 discusses some new ideas that may be incorporated to enhance the reliability of the OCR process.
Chapter 2

Optical Character Recognition models

In this chapter, models for optical character recognition are described. We demonstrate how the process works, as well as how the errors are modeled.

2.1 Pattern-matching model

Probably the most common model for OCR is the pattern-matching model. For each uppercase and lowercase letter, number, and punctuation mark, a detailed image is stored in the OCR engine. Each image is represented as an arrangement of black and white pixels. After a physical paper document is scanned, the OCR engine attempts to interpret the image of the scanned document letter by letter. One simple method would be to superimpose the image of each letter onto the first letter in the document and determine which one fits the best. The letter that fits the best would be the one that differs from the one on the page by the fewest pixels. Then the OCR tries to find the next letter that would best fit the next character on the page. The procedure continues for the entire line and ultimately the entire page.

One common way to model the error rates for OCR based on pattern matching is to build a confusion matrix. For every character whose image is stored in the OCR engine, the confusion matrix will have a row and column corresponding to that
character. Entry \((i,j)\) represents the probability that character \(i\) would be interpreted as character \(j\). To compute the confusion matrix, one needs to know the probability that a given pixel in an image will flip value. This change in value could occur while the image is scanned, or it could occur from a corruption in the original document. To simplify the computation of the confusion matrix, we assume that all pixels have the same probability of flipping, and that each pixel is independent of all other pixels. Another simplifying assumption is that black pixels change into white pixels at the same rate as white pixels flip to black ones. We let \(p\) denote this probability.

With these assumptions, we can compute the probability that enough pixels have flipped to make character \(i\) look more like character \(j\). To do this, we need to compute the Hamming distance between the images of those two characters. We position the images so that the baselines are aligned and the images are centered upon each other. (We then try to maximize the number of pixels that share the same value in both images.) Then we count the number of pixels that have different values between the two images. This number is the Hamming distance between the two images.

We presume that in order for character \(a\) to be mistaken for \(b\), the number of pixels needed to flip must be at least half the Hamming distance from \(a\) to \(b\). If \(h\) is the Hamming distance, then the probability that \(a\) will look more like \(b\) is

\[
\sum_{i = [h/2]}^{h} \binom{h}{i} p^i (1 - p)^{h - i}.
\]

If \(h\) is large, we could also approximate this probability with the central limit theorem since these pixel flips represent a Bernoulli process. The total number of pixel flips is a binomial random variable whose distribution tends to a normal distribution for large \(h\).

One property from our model is that the confusion matrix is symmetric. This reflects the symmetry of the Hamming distance function and the single probability for a pixel to change value.

Of course, this model does not reflect the error rates for an OCR engine perfectly. The assumptions we made to simplify computations are probably not true. First,
the pixels near the boundary of a character (i.e. a pixel surrounded by pixels of both colors) might have a higher chance of flipping than pixels in the character's interior or exterior. Furthermore, pixel changes might not be independent of each other. That is, if one pixel changes value, then the surrounding pixels are likely to have a higher probability of changing as well. And finally, white pixels might turn black more often than black pixels turn white.

Another drawback to this model is that letters might get split into two letters. For instance, the letter $m$ could be misinterpreted as a cluster $rn$. Similarly, it is possible for two letters to fuse into one; the cluster $rn$ could just as well be read as $m$. This model also does not track characters that are omitted or added during the OCR process.

A third drawback is that if enough pixels are corrupted, there may be several characters with which the corrupted image may have more pixels in common than with the original character itself. Therefore, each entry in the confusion matrix is theoretically overestimated. However, the probabilities are going to be very small anyway, so this issue will probably have little effect on the model.

2.2 Document Image Decoding

Document Image Decoding (DID) is one model of optical character recognition based on pattern matching [2]. The method of DID is patterned after the hidden Markov models used in speech recognition. The DID model of the OCR problem involves an image source, a channel, and a decoder. A message source generates messages with a certain probability distribution. The imager converts the message into an ideal image consisting of pixels with binary values. Together, the message source and imager make up the image source. Next the image is fed through a channel. In this channel, distortions, blurring, and noise will occur. Finally, the decoder receives the new image from the channel and tries to reconstruct the original message.

In DID, several problems of theoretical interest arise. The decoding problem is often interpreted as a graph theoretic problem in which a path must be found so that
a normalized probability of the path is maximized. Shortest path algorithms and
dynamic programming become important techniques in DID.

The DID method is useful for simulating a typical OCR process. With an imple-
mentation of DID, real data can be gathered on what errors occur most often. After
determining why they occur, the assist channel can be used to concentrate on those
events and correct them or prevent them from happening.

2.3 Other OCR models

Another way to do OCR is to analyze each character topologically. The OCR engine
would try to detect special features such as closed loops, straight or curved strokes,
points of intersection, and angles. Thus capital B would be recognized from its two
closed loops. The letter x has a central intersection where four segments join. Subtle
differences must be detected to distinguish similar letters. Capital O is made up from
one single circular loop, while capital D is a loop made from a straight line and a
semicircle. The weakness in this model is that uppercase and lowercase letters such
as C and c would often be confused with each other.

Finally, some OCR engines use lexicons to decipher words. In some cases, lexicons
can be useful in determining an unclear letter from its surrounding letters.
Chapter 3

Separation Coding

3.1 Purpose and Description

One common problem in optical character recognition is the confusion between similar characters. For example, the letters c, e, and o are all small, round letters that differ by a stroke. Perhaps a stronger example is the period and the comma, since they differ by a few pixels.

In order to reduce the rate of confusing letters that look like one another, we employ a technique we call separation coding. This technique is basically a version of data compression. To form a separation code, we partition the set of characters into subsets. We would like this partition this set in a way so that characters that look similar would go into different subsets.

Each subset is assigned a label that may consist of a group of bits. The assist channel would then be a “translation” of the characters in the primary channel to the subsets labels of the separation code. Thus, if the letters c, a, and r are in groups 11, 00, and 01, respectively, then the word car in the primary channel would correspond to the sequence 110001 in the assist channel.

With the aid of separation coding, the OCR engine would be able to catch many of its errors by detecting inconsistencies between the primary channel and the assist channel. For instance, suppose that just as in the previous example, car corresponds to 110001, and the word car appears in the document. Now suppose also that the
engine misinterpreted the c as an e. If the letters c and e were thoughtfully placed in separate subsets (11 and 10, for instance), the OCR engine would detect the mistake and then try to correct it. This time, on its second guess, the engine will know that the correct letter must come from the subset labeled 11. The engine will then attempt to find a similar looking letter from that subset and recognize the correct letter to be c.

3.2 Design

When designing a separation code, maximizing accuracy is not the only goal. We also want to use as little space as possible. After all, we could put each character into its own group, but then each group would require a label of perhaps seven bits. We would essentially have an exact ASCII representation in the assist channel, so there would be no compression.

To design a separation code, we need to compute the probabilities that character i turns into character j. Certainly the confusion matrix described in the previous chapter is needed to determine which characters look similar. However, it is not enough to base the separation code on the confusion matrix alone. After all, there may be characters that look similar to each other, but are rarely used (such as \ and |). For those kinds of characters, it is not so crucial for them to be separated. Therefore, we also need to know the character frequencies in order to design a separation code.

The (i, j)-th entry of a confusion matrix represents the probability that character i changes to j, given that the character on the page is i. We multiply this conditional probability by the probability that a randomly selected character on a page is character i. This product, which we shall call the confusion frequency from i to j, is the probability that upon reading some character, it had interpreted i as j. These are the probabilities we want to use in building the separation code.
3.3 Building Algorithm for the Separation Code

Let us consider the problem of reducing the error rate with a separation code of \( n \) groups. This problem becomes especially important when we want to incorporate error-correction techniques along with separation codes.

It is in fact possible to guarantee a reduction by a factor of \( n \). We can represent our model first as a directed graph in which the vertices represent characters, and each edge \((u, v)\) is weighted with the confusion frequency from \( u \) to \( v \). However, it is not necessary to keep track of edge directions; we can replace edges \((u, v)\) and \((v, u)\) with an undirected edge on \((u, v)\). The weight of this edge is the sum of the confusion frequencies from \( u \) to \( v \) and \( v \) to \( u \). We then group the nodes of this complete weighted graph so that we minimize the sum of the weights of edges between nodes within the same group. We describe that algorithm:

1. First, start out with \( n \) empty groups.

2. For each character \( c \) (represented as a node),
   
   (a) First add it to the graph along with all edges connecting it to all other nodes in the graph.
   
   (b) For each group, sum up the new edges connecting the elements in that group to the new character.
   
   (c) Add \( c \) to the group for which the additional weight is minimized.

We prove that the algorithm can guarantee a reduction by at least a factor of \( n \). This is done by induction on the number of nodes. When the first node is placed, the total error is 0, which is better than a \( 1/n \) reduction. Now suppose there are \( k \) nodes in the graph, and that the inductive hypothesis is true. When we add another node and the edges connecting it to the existing nodes, we increase the total weight of the graph by \( w \). The average increase per group is \( w/n \). So for at least one group, adding the \((k + 1)\)st node to that group increases the weight by no more than \( w/n \).
Thus the total inter-group error rate remains less than $1/n$ of the sum of the edge weights.

Although the algorithm above guarantees the reduction, it does not generate an optimal separation code. Finding an optimal separation code is an NP-complete problem, for it is a generalization of the Max-cut problem [1]. Although it is NP-complete, we can settle for a search algorithm, which works as follows:

1. Randomly place the characters into groups.

2. For each character $c$, let $\text{gerr}[c,g]$ represent the sum of the arcs that connect node $c$ to the nodes in group $g$.

3. Set $\text{reduction}=0$.

4. If $c$ is in group $g_1$, but $\text{gerr}[c,g_2] < \text{gerr}[c,g_1]$ for some other group $g_2$, then

   a) $\text{reduction}$ increases by $\text{gerr}[c,g_1] - \text{gerr}[c,g_2]$.

   b) Move $c$ into $g_2$.

   c) For each character $d$ (different from $c$),

      i. $\text{gerr}[d,g_1] = \text{gerr}[d,g_1] - \text{err}[c,d]$ (where $\text{err}[c,d]$ is the weight of arc $cd$)

      ii. $\text{gerr}[d,g_2] = \text{gerr}[d,g_2] + \text{err}[c,d]$.

5. Repeat the previous step until no further reductions can be made.

    While it is not entirely necessary, in step 4, we can first compute which character $c$ will enable the largest reduction. This would essentially make this algorithm a greedy one.

    To try to find the most nearly optimal separation code, we can run many trials. Each individual trial can guarantee a reduction by a factor of $1/n$. However, in practice, the reduction is usually much greater.
Chapter 4

Error Correcting Codes

4.1 Introduction

In this chapter, we introduce the well-known technique of error correcting codes. Error-correcting codes are widely used for transmission of data across a network. The sequence of bits is divided into blocks. Each block contains not only information but also several check digits. The purpose of the check digits is to ensure that the information has been transmitted perfectly. If the transmission is perfect, the block (including its check digits) will satisfy some mathematical relation. Those types of blocks are also called code words.

The $t$-error correcting code is designed to be able to correct up to $t$ errors within a block. These errors may occur in any part of the block, including the check digits. If $t$ or fewer errors occurred in the transmission of a block, then the error-correcting algorithm will locate the nearest code word in Hamming distance (i.e. that differs by the fewest bits). Thus any two different acceptable code words must differ by at least $2t + 1$ bits.

There are many variations of error-correcting codes. One such variant is known as Bose-Chaudhuri-Hocquenghem (BCH) codes. The block length of a BCH code is $n = 2^m - 1$ bits for some integer $m$, and the number of check-digits is at most $mt$. Typically Galois fields are the basis for the implementation of a BCH code.

Binary BCH codes can be generalized to $q$-ary BCH codes, where $q$ is the power
of some prime $p$. Thus a block will consist of digits rather than bits, and there are $q$ such digits instead of 2. One special case of BCH codes is known as the Reed-Solomon codes. While $q$-ary BCH codes have blocks of length $q^s - 1$ for some positive integer $s$, Reed-Solomon codes have blocks of length $q - 1$. A $t$-error-correcting Reed-Solomon code will have $2t$ parity digits per block, and minimum distance of $2t + 1$ digits.

The main foundation of error correction is finite field arithmetic. To correct the block of errors, it first converts the digits of the block into a polynomial. If the block has no errors, this polynomial would be a multiple of a certain generator polynomial. Otherwise, the error location polynomial is computed by an algorithm due to Berlekamp, and the roots of this polynomial tell where the errors lie. Finally, the correct characters must be computed by using another formula. The full algorithms for error correcting codes can be found in [3].

4.2 Use of Error Correction in OCR

In this chapter, we demonstrate how Reed-Solomon codes are used in the assist channel. The primary channel, i.e. main text on the page, is divided vertically into columns. These columns are then grouped into blocks. The corresponding check digits for each block are placed in the assist channel.

We will use the term segment to denote one line within a block. Thus if a block has $k$ columns, then a segment of the block consists of $k$ characters.

We describe the correction process in this version of assist channel coding. First, the OCR engine produces its interpretation of the document. It then partitions the document into blocks of the appropriate size (which is also to be specified in the assist channel). In this case, let each block contain two columns. The first two characters in each line would be included in the leftmost block. (There is an issue about how to deal with spaces, but that will be discussed later.) The algorithm for Reed-Solomon codes determines whether the OCR engine committed any errors. If there are, the correct characters replace the erroneous ones, and the OCR engine uses the correct information to reinterpret each line that had contained an error. Reinterpretation is
necessary in case the error reflects an added or deleted character. Once all the lines are reinterpreted, the third and fourth characters comprise the next block, and it is checked for errors. Once again, after errors are corrected, the lines with errors are reinterpreted, knowing the first two blocks (four characters) are correct. This process is repeated until the end of the page is reached.

Note that synchronization is important in a Reed-Solomon code. If a character were added to or deleted from a line, the following characters would then shift over one place. Thus one simple mistake would propagate into many errors, possibly exceeding the error-correcting capacity of the code. This problem would be especially bad if we had divided the page horizontally so that each block contains one or more lines of text. Thus, to avoid this synchronization problem, each block instead consists of one or more columns.

This brings us to the next problem: how many columns should each block contain? To keep the design simple, each block should contain the same number of columns (except for possibly the last block in the page, since the number of characters varies from line to line). The optimal number of columns will depend on the rates of two types of OCR errors: replacement errors and synchronization errors. A replacement error is one in which one character is merely substituted for another; for instance, if the OCR engine interprets the word *hall* as *hail*, then it has made a replacement error by substituting an *i* for an *l*. Meanwhile, a synchronization error is one in which a character has been added or deleted. Examples include *pint* turning into *print* (adding an *r*) or into *pit* (deleting the *n*).

If replacement errors happen much more frequently than synchronization errors, we could expect larger block sizes to work better. The reason is that as the block size increases, there would be a decrease in the probability that the number of errors will exceed the strength of the error correction. In a fixed width font such as Courier, we could expect the OCR engine to make very few synchronization errors since the characters are aligned perfectly. However, if synchronization errors happen much more frequently than replacement errors, then large block sizes may not be such a good idea. If a synchronization error happened in the left half of a segment, then the
rest of the characters in the segment would shift over and become incorrect. Thus smaller block sizes would work better in this case.

We compare block sizes by computing the failure rates of a page. A page failure occurs when a single block fails. Block failures are based on how much the error correction strength of the block exceeds the expected number of errors in a single block. No matter the block size, we hold the size of the page (in lines and characters per line) and total error correction strength (i.e. check digits in the assist channel) constant. The optimal block size is the one with the smallest page failure rate.

4.3 Models for Error Simulation

There are several models we can use to simulate replacement and synchronization errors in the OCR of a page of text. We shall describe two of them.

We compute the probability distribution on the number of errors in a segment by deriving a generating function that carries these probabilities in its coefficients. These generating functions are then used to compute expected numbers of errors in blocks of various sizes, and their respective standard deviations.

4.3.1 One desynchronization error per row

The first model assumes that once a synchronization error is made in a line of text, the rest of the characters in the line become errors. Let \( r \) be the probability that a replacement error occurs at any point, and \( s \) be the probability that a synchronization error occurs. Let \( k \) be the number of columns in a block. We can calculate the probability \( p_{ik} \) that \( i \) errors occur in one segment of the block. The probability that the number of errors \( i = 0 \) is \((1 - r - s)^k\). For other values of \( i \), we express it as a sum of different cases based on where the synchronization error happens, if it does. Let’s say that \( j \) of the errors are replacement errors, and that a synchronization error causes the other \( i - j \) errors. Then the replacement errors must occur only within the first \( k - (i - j) = k - i + j \) positions, and each happened with probability \( r \). (They cannot occur in the final \( i - j \) positions, since no replacement errors follow
desynchronizations.) These replacement errors can be arranged in \( \binom{k-i+j}{j} \) ways. In any case, \( k - i \) characters have been correctly read. And finally, a synchronization error occurred once unless \( j = i \). Thus the total probability that \( i \) errors occurred is

\[
(1 - r - s)^{k-i} \left( \binom{k}{i} r^i + \sum_{j=0}^{i-1} \binom{k-i+j}{j} r^j s^j \right).
\]

We next find the generating function of two variables \( z \) and \( w \) such that the coefficient \( p_{ik} \) of \( z^i w^k \) is the probability that there are \( i \) errors within a segment of length \( k \). The generating function, which we call \( G(z, w) \), satisfies the following equation:

\[
G - \frac{1}{1 - (1 - r - s)w} = w(1 - r - s)(G - \frac{1}{1 - (1 - r - s)w}) + rzwG + s \frac{1}{1 - wz} - s.
\]

The explicit formula becomes

\[
G(z, w) = \frac{1 - wz + swz}{(1 - wz)(1 - w + rw + sw - rwz)}.
\]

The drawback of this model is that it may be hard to distinguish synchronization errors from a run of replacement errors. Suppose that instead of the correct string “abcdefg”, we get “abceffg” from OCR. Is the mutation from “de” to “ef” a result of two different replacements, d to e and e to f, or was the letter d dropped and an extra letter f added during the OCR process? To resolve this and similar ambiguities, we introduce the next model based on runs of errors.

### 4.3.2 Runs of Errors

In this model, we take into account the possibility that two desynchronizations within the same line may cancel each other out. Imagine an infinitely long line of text. We will assume that each run of incorrect characters is finite; that is, the probability that we get an infinitely long run is negligibly small.

Our goal is to find a generating function \( K(z, w) \) for which the coefficient of \( z^i w^k \) is the probability that \( i \) errors occur in a segment, given that the segment has length
This information is essential for making design decisions. We use the following step-by-step derivation to find \( K(z, w) \).

First we let \( p_i \) stand for the probability that a character will be the beginning run of \( i \) incorrect characters followed by a correct one, given that the previous character was read correctly. With this notation, \( p_0 \) actually represents the probability the character is read correctly, given that the previous character has been also read correctly. Then \( P(x) \) is the generating function whose coefficients are \( p_i \).

\[
P(x) = \sum_{i=0}^{\infty} p_i x^i
\]

Note that \( P(1) = 1 \). The advantage of using \( P(x) \) is that run lengths are easy to measure and runs are easy to tally. We avoid the ambiguity that may arise when we try to count replacement and synchronization errors (as in \( cdef \) changing to \( ceff \)).

We next analyze the special case in which no run starts within a segment and continues into the next segment. Thus we define \( G(z, w) \) to be the generating for which each coefficient \( g_{ik} \) of \( z^i w^k \) represents the probability that in one segment of length \( k \), there are \( i \) errors, and that the final character in that segment is correct. In other words, the run of incorrect characters does not go beyond the current block and into the next block.

We show that by successively adding powers of \( wP(wz) \), we produce \( G(z, w) \). First, \( wP(wz) \) represents the fact that for a segment of length \( k + 1 \), there is exactly one way for the maximum number of errors \((k)\) to occur: the first \( k \) are incorrect, followed by the remaining correct character. The probability that this maximum occurs is \( p_k \). As we raise \( wP(wz) \) to the \( n \)th power, we produce the probabilities that the block of size \( k + n \) is divided into \( n \) runs (note that a string of zero incorrect characters followed by a correct character counts as a run). This happens because each \( wP(wz) \) contributes one run to the block. Finally, summing the powers of \( wP(wz) \), we combine all the possible subdivisions of a block into runs. Note that no two different powers of \( wP(wz) \) have any terms \( z^i w^k \) in common. Thus,
\[ G(z, w) = \sum_{k=0}^{\infty} \sum_{j=0}^{k} g_{ik} z^i w^k = \sum_{n=0}^{\infty} w^n P(zw)^n = \frac{1}{1 - wP(zw)}. \]

The first few terms in this series are:

\[
1 + p_0 w + (p_0^2 + p_1 z) w^2 + (p_0^3 + 2p_0 p_1 z + p_2 z^2) w^3 + \\
(p_0^4 + 3p_0^2 p_1 z + 2p_0 p_2 z^2 + z^2(p_1^2 + p_2 z)) w^4 + o(w^5)
\]

\( G(z, w) \) is close to \( K(z, w) \), but it does not include the case in which a desynchronization extends beyond the segment. In order to include this case, we need to compute the generating function \( H(x) \) whose coefficients \( h_i \) for \( x^i \) represent the probability that the run of incorrect characters is at least \( i \). The coefficients of \( x^i \) must be \( \sum_{n=i}^{\infty} p_n \), or \( 1 - \sum_{n=0}^{i-1} p_n \).

\[
H(x) = \sum_{i=0}^{\infty} h_i x^i = 1 + \frac{(1 - P(x))x}{1 - x} = 1 + (1 - p_0)x + (1 - p_0 - p_1)x^2 + (1 - p_0 - p_1 - p_2)x^3 + \\
(1 - p_0 - p_1 - p_2 - p_3)x^4 + o(x^5)
\]

Consider what happens when we substitute \( zw \) for \( x \) above. The coefficients of \( z^i w^i \) in \( H(zw) \) represents the probability that a run of at least \( i \) error occurs on the first character in a segment that is \( i \) columns wide. For the block, that line would have exactly \( i \) errors.

Finally, to get the desired generating function \( K(z, w) \), we multiply \( G(z, w) \) by \( H(zw) \). Let’s interpret what the coefficients represent in \( G(z, w)H(zw) \). Each coefficient of \( z^i w^k \) is the sum of products of coefficients in \( G(z, w) \) and \( H(zw) \). Thus we can imagine splitting up a segment of \( k \) characters with \( i \) errors into two components. The first component goes from the beginning of the block up to (and including) the last correct character in that segment. Let \( k - j \) be the length of the first component. The second component consists of the remaining incorrect characters to the end of the segment. Thus the second component has length \( j \) and consists of \( j \) errors since
that component must consist entirely of errors. (Note that \( j \) cannot exceed \( i \).) That leaves \( i - j \) errors in the first component.

We have defined \( g_{(i-j),(k-j)} \) to be the probability that the first component has \( i - j \) errors. In addition, \( h_j \) is the probability that the second component has no less than \( j \) errors. (It could be more, but we only care about the first \( j \) errors of that run.) So the product \( g_{(i-j),(k-j)} h_j \) is the probability that the original \( k \)-character segment has \( i \) errors, of which the final \( j \) errors constitute the second component. And from the multiple ways of dividing the segment of components, the total probability that a segment of length \( k \) has \( i \) errors is \( \sum_{j=0}^{i} g_{(i-j),(k-j)} h_j \). This is the coefficient of \( z^i w^k \) in \( K(z, w) \).

The explicit formula for \( K(z, w) \) is

\[
K(z, w) = G(z, w)H(zw) = \frac{1}{1 - wP(zw)} \left( 1 + \frac{zw(1 - P(zw))}{1 - zw} \right) = \frac{1 - zwP(zw)}{(1 - wP(zw))(1 - zw)}.
\]

We show the first few terms in the series of \( K(z, w) \):

\[
K(z, w) = G(z, w)H(zw) = 1 + (p_0 + (1 - p_0)z)w + (p_0^2 + ((1 - p_0)p_0 + p_1)z + (1 - p_0 - p_1)z^2)w^2 + (p_0^3 + ((1 - p_0)p_0^2 + 2p_0p_1)z + (p_0(1 - p_0 - p_1) + (1 - p_0)p_1 + p_2)z^2 + (1 - p_0 - p_1 - p_2)z^3)w^3 + (p_0^4 + ((1 - p_0)p_0^3 + 3p_0^2p_1)z + (p_0^2(1 - p_0 - p_1) + 2(1 - p_0)p_0p_1 + p_1^2 + 2p_0p_2)z^2 + ((1 - p_0 - p_1)p_1 + p_0(1 - p_0 - p_1 - p_2) + (1 - p_0)p_2 + p_3)z^3 + (1 - p_0 - p_1 - p_2 - p_3)z^4)w^4 + o(w)^5
\]

The coefficient of \( w^k \) in \( K(z, w) \) is a generating function whose coefficients of \( z^i \) are probabilities that sum up to 1. Thus as a sanity check, we verify that setting
z = 1 produces the generating function $\sum_{n=1}^{\infty} w^n$, i.e. $(1 - w)^{-1}$:

$$K(1, w) = \frac{1 - wP(w)}{(1 - wP(w))(1 - w)} = \frac{1}{1 - w}.$$  

### 4.4 Computing Statistical Functions

We have just developed the generating function $K(z, w)$ in which the coefficient $p_{ik}$ of $z^iw^k$ is the probability that $i$ errors occur in a segment of a block, given that the block has $k$ columns. Now we show how to compute the expected number of errors in a block of size $k$, for all $k \geq 0$. This is done simply by taking the partial derivate of $K$ with respect to $z$ and setting $z = 1$. If we let

$$K = \sum_{k=0}^{\infty} \sum_{i=0}^{\infty} p_{ik}z^iw^k$$

then

$$\frac{\partial K}{\partial z} = \sum_{k=0}^{\infty} \sum_{i=0}^{\infty} ip_{ik}z^{i-1}w^k$$

$$\left. \frac{\partial K}{\partial z} \right|_{z=1} = \sum_{k=0}^{\infty} \sum_{i=0}^{\infty} ip_{ik}w^k$$

But note that $\sum_{i=0}^{\infty} ip_{ik}$ is the expected number of errors in a block of size $k$. The generating function clearly generates these expected values.

To compute the variance, we use the well-known identity $\text{Var}[X_k] = E[X_k^2] - E[X_k]^2$, where $X_k$ is the random variable of the number of errors in a block. Thus we need a method to compute $E[X_k^2]$, that is, the second moment. We can accomplish this task with the following derivation:

$$\frac{\partial^2 K}{\partial z^2} + \frac{\partial K}{\partial z} = \sum_{k=0}^{\infty} \sum_{i=0}^{\infty} i(i - 1)p_{ik}z^{i-2}w^k + \sum_{k=0}^{\infty} \sum_{i=0}^{\infty} ip_{ik}z^{i-1}w^k$$

$$= \sum_{k=0}^{\infty} \sum_{i=0}^{\infty} i(i - 1)p_{ik}z^{i-2}w^k + ip_{ik}z^{i-1}w^k$$

27
These generating functions produce all the values that enable us to compute the statistical values. We can use Mathematica to compute the series expansion of each generating function and extract their coefficients to produce $E[X_k]$ and $E[X_k^2]$. Then we can compute $\text{Var}[X_k] = E[X_k^2] - E[X_k]^2$.

4.5 Comparing Block Sizes

There are two approaches that can be taken to compute optimal block size, given the probability distribution on the length of runs of errors. The first is to compute the failure rates from the block sizes (as well as error-correction strength). The other approach is to compute the minimum error-correction strength necessary for each block size to achieve a particular failure rate. The second approach is basically an inverse of the first one.

4.5.1 Failure rates

The value we want to compute from the first approach is the rate that the error correction fails for a page. A page failure occurs whenever the number of errors within any block on that page exceeds the error-correction strength. (Such a failure is called a block failure.) It is presumed that once a block fails to be corrected, the following blocks will be severely affected. Thus for OCR to succeed on a page, all the blocks must be corrected to perfection.

In order to compute the page failure rate, we must compute the block failure rate. We can imagine a block consisting of $n$ independent rows. For each row $i$, we assign a random variable $X_i$ of the number of errors in that row. The sum $S_n$ of these random variables is in the random variable of the number of errors in the entire block. We then use the Central Limit Theorem to estimate the probability that the number
of errors in the whole block exceeds the error-correction strength. Recall that the
Central Limit Theorem states

$$\lim_{n \to \infty} \left[ Pr \left( \frac{S_n - n\mu}{\sigma \sqrt{n}} \geq y \right) \right] = Q(y)$$

where we define

$$Q(y) = 1 - \text{erf}(x/\sqrt{2})$$

$$= \int_{y}^{\infty} \frac{1}{\sqrt{2\pi}} \exp(-x^2/2) dx.$$ 

In this formula, $\mu$ is the expectation of $X_i$, and $\sigma$ is the standard deviation. The
variable $y$ is the number of standard deviations of $S_n$ (i.e. $\sigma \sqrt{n}$) above the mean $n\mu$.

Let $k$ be the number of columns per block, $E$ be the error-correction strength (i.e. the
number of errors per column that the assist channel can handle), and $n$ be the
number of lines. Then $kE$ is the maximum number of errors than can be corrected
in a block. The block failure rate is the probability that $S_n \geq kE$. We solve for $y$ and
conclude that the formula for the block failure rate $B$ is

$$B = Q \left( \frac{kE - n\mu}{\sigma \sqrt{n}} \right).$$

Now let $c$ be the number of columns in the entire page. There are $c/k$ blocks in
a page. Once we have $B$, we find the page failure rate $P$ with the equation

$$P = 1 - (1 - B)^{(c/k)}.$$ 

The quantity $1 - P$ is the page success rate. The page is read successfully only when
all $c/k$ blocks are read correctly.

### 4.5.2 Required error-correction strength

As a useful alternative calculation, we show how to compute the amount of error
correction required to meet a specific page failure rate. The equations are basically
<table>
<thead>
<tr>
<th>$i$</th>
<th>$p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.998025</td>
</tr>
<tr>
<td>1</td>
<td>0.000744236</td>
</tr>
<tr>
<td>2</td>
<td>0.0000748916</td>
</tr>
<tr>
<td>3</td>
<td>0.00000889338</td>
</tr>
<tr>
<td>4</td>
<td>0.00000748916</td>
</tr>
<tr>
<td>5</td>
<td>0.0000187229</td>
</tr>
<tr>
<td>6</td>
<td>0.0000374458</td>
</tr>
<tr>
<td>7</td>
<td>0.0000421265</td>
</tr>
<tr>
<td>8</td>
<td>0.0000234036</td>
</tr>
<tr>
<td>9</td>
<td>0.000800404</td>
</tr>
</tbody>
</table>

Table 4.1: The probabilities of runs of length $i$ occurring (original documents).

the inverse operations of the ones in the previous subsection.

$$B = 1 - (1 - P)^{(k/c)}$$

$$E = n\mu + \sigma\sqrt{nQ^{-1}(B)}$$

### 4.6 Sample Calculation

The following example shows how the theory developed in the previous sections can be applied to actual data. This example takes data from OCR on an ordinary batch of legal documents fresh from the printer.

Table 4.1 shows the probability distribution of the length of the error runs. Note, however, that we let $p_{10}$ be the number of runs of length 10 or greater.

We compute the series for $K(z, w)$, up to $z^4 w^4$:

$$K(z, w) =$$

$$1 + (0.998025 + 0.00197527 z) w + \left(0.996053 + 0.0027156 z + 0.00123103 z^2\right) w^2 +$$

$$\left(0.994086 + 0.003453 z + 0.00130496 z^2 + 0.00115614 z^3\right) w^3 +$$

$$\left(0.992122 + 0.00418748 z + 0.00137915 z^2 + 0.00124385 z^3 + 0.00106721 z^4\right) w^4$$

30
Table 4.2: Error statistics for different segment lengths (original documents)

The series of means, which is \( \partial K/\partial z \) evaluated at \( z = 1 \), becomes

\[
0.00197527 w + 0.00517766 w^2 + 0.00953134 w^3 + 0.0149462 w^4 + \\
0.0213463 w^5 + 0.0287121 w^6 + 0.0369726 w^7 + 0.0460897 w^8
\]

The series of second moments becomes

\[
0.00197527 w + 0.00763973 w^2 + 0.0190781 w^3 + 0.0379741 w^4 + \\
0.0657811 w^5 + 0.104271 w^6 + 0.154467 w^7 + 0.217608 w^8
\]

The variances are then computed by taking the second moments and then subtracting the squares of the means. The standard deviations are simply the square roots of the variances. We summarize all the values in Table 4.2.

Next, we compute the block and page failure rates. To keep things simple, we fix the number of rows and columns on a page. Let the page consist of 40 lines and 100 columns. Table 4.3 shows the block failure rates \( (B) \) versus block size and error-correction strength. Table 4.4 shows page failure rates \( (P) \) compared to block size and error-correction strength.

Finally, we show probably the most useful table of all: required error-correction strength needed to achieve a specific page failure rate, for different block sizes (Table 4.5). Once again, we set the number of rows to 40 and columns to 100.

From the tables of page failure rates and required error-correction strength, it is
<table>
<thead>
<tr>
<th>k</th>
<th>(0.5)</th>
<th>1</th>
<th>1.5</th>
<th>2</th>
<th>2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0669118</td>
<td>0.00051949</td>
<td>2.09286 (\times 10^{-7})</td>
<td>3.93569 (\times 10^{-12})</td>
<td>0.</td>
</tr>
<tr>
<td>2</td>
<td>0.0753813</td>
<td>0.00057907</td>
<td>2.08373 (\times 10^{-7})</td>
<td>3.13677 (\times 10^{-12})</td>
<td>0.</td>
</tr>
<tr>
<td>3</td>
<td>0.0996199</td>
<td>0.00132827</td>
<td>1.1443 (\times 10^{-6})</td>
<td>5.69279 (\times 10^{-11})</td>
<td>1.66533 (\times 10^{-15})</td>
</tr>
<tr>
<td>4</td>
<td>0.126925</td>
<td>0.00281481</td>
<td>5.50885 (\times 10^{-6})</td>
<td>8.52193 (\times 10^{-10})</td>
<td>9.9365 (\times 10^{-15})</td>
</tr>
<tr>
<td>5</td>
<td>0.154255</td>
<td>0.00515998</td>
<td>0.000019656</td>
<td>7.65494 (\times 10^{-9})</td>
<td>2.91045 (\times 10^{-13})</td>
</tr>
<tr>
<td>6</td>
<td>0.181337</td>
<td>0.00853932</td>
<td>0.0000567368</td>
<td>4.78734 (\times 10^{-8})</td>
<td>4.89969 (\times 10^{-12})</td>
</tr>
<tr>
<td>7</td>
<td>0.207046</td>
<td>0.0128393</td>
<td>0.000133505</td>
<td>2.09964 (\times 10^{-7})</td>
<td>4.7714 (\times 10^{-11})</td>
</tr>
<tr>
<td>8</td>
<td>0.231321</td>
<td>0.0179985</td>
<td>0.000270672</td>
<td>7.11076 (\times 10^{-7})</td>
<td>3.11831 (\times 10^{-10})</td>
</tr>
</tbody>
</table>

Table 4.3: Block failure rates for original set of documents

<table>
<thead>
<tr>
<th>k</th>
<th>(0.5)</th>
<th>1</th>
<th>1.5</th>
<th>2</th>
<th>2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.999018</td>
<td>0.0506357</td>
<td>0.0000209284</td>
<td>3.93574 (\times 10^{-10})</td>
<td>0.</td>
</tr>
<tr>
<td>2</td>
<td>0.980133</td>
<td>0.0285465</td>
<td>0.0000104186</td>
<td>1.56841 (\times 10^{-10})</td>
<td>0.</td>
</tr>
<tr>
<td>3</td>
<td>0.96974</td>
<td>0.0433381</td>
<td>0.0000381428</td>
<td>1.8976 (\times 10^{-9})</td>
<td>7.32747 (\times 10^{-15})</td>
</tr>
<tr>
<td>4</td>
<td>0.966404</td>
<td>0.0680438</td>
<td>0.000137712</td>
<td>2.13048 (\times 10^{-8})</td>
<td>2.498 (\times 10^{-13})</td>
</tr>
<tr>
<td>5</td>
<td>0.964942</td>
<td>0.0982941</td>
<td>0.000393046</td>
<td>1.53099 (\times 10^{-7})</td>
<td>5.8201 (\times 10^{-12})</td>
</tr>
<tr>
<td>6</td>
<td>0.964389</td>
<td>0.133188</td>
<td>0.000945194</td>
<td>7.9789 (\times 10^{-7})</td>
<td>8.16605 (\times 10^{-11})</td>
</tr>
<tr>
<td>7</td>
<td>0.963635</td>
<td>0.168568</td>
<td>0.00190553</td>
<td>2.99949 (\times 10^{-6})</td>
<td>6.81629 (\times 10^{-10})</td>
</tr>
<tr>
<td>8</td>
<td>0.962691</td>
<td>0.203103</td>
<td>0.00337814</td>
<td>8.88841 (\times 10^{-6})</td>
<td>3.89789 (\times 10^{-9})</td>
</tr>
</tbody>
</table>

Table 4.4: Page failure rates for original set of documents

<table>
<thead>
<tr>
<th>k</th>
<th>(10^{-3})</th>
<th>(10^{-4})</th>
<th>(10^{-5})</th>
<th>(10^{-6})</th>
<th>(10^{-7})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.27661</td>
<td>1.41382</td>
<td>1.53904</td>
<td>1.65492</td>
<td>1.76326</td>
</tr>
<tr>
<td>2</td>
<td>1.23684</td>
<td>1.3759</td>
<td>1.50216</td>
<td>1.61856</td>
<td>1.72709</td>
</tr>
<tr>
<td>3</td>
<td>1.29276</td>
<td>1.44198</td>
<td>1.57701</td>
<td>1.70122</td>
<td>1.81683</td>
</tr>
<tr>
<td>4</td>
<td>1.36118</td>
<td>1.5212</td>
<td>1.66564</td>
<td>1.79829</td>
<td>1.92158</td>
</tr>
<tr>
<td>5</td>
<td>1.42855</td>
<td>1.59882</td>
<td>1.75222</td>
<td>1.89288</td>
<td>2.0235</td>
</tr>
<tr>
<td>6</td>
<td>1.49532</td>
<td>1.67554</td>
<td>1.83761</td>
<td>1.98606</td>
<td>2.12378</td>
</tr>
<tr>
<td>7</td>
<td>1.55751</td>
<td>1.74693</td>
<td>1.91703</td>
<td>2.07267</td>
<td>2.21696</td>
</tr>
<tr>
<td>8</td>
<td>1.61578</td>
<td>1.81378</td>
<td>1.99136</td>
<td>2.15369</td>
<td>2.30409</td>
</tr>
</tbody>
</table>

Table 4.5: Required error-correction strength for original documents
clear that the optimal block size is 2 for this set of documents. One column per block is the next optimal block size, followed by 3 columns per block. (Note, however, that when the error correction strength is at a weak 0.5 errors per column, the optimal block size is at least 8, as shown on Table 4.4. But no one wants the page failure rate to be 96 percent.)
Chapter 5

Experiments and Results

5.1 Data-gathering procedure

Once the theoretical models have been developed, experiments were done to test these models. In order to test the models, we needed to gather data on character frequency and OCR error rates. Sixty legal documents of varying length, totaling 120 pages, were used as data. Additional data were also drawn from news reports.

The goal was to design the assist channel to be robust so that it could withstand the transformations of photocopying a document several times. That is, after copying the document, we make a (secondhand) copy of that copy, and then create a (third-hand) copy of the secondhand copy, and so forth. Each copy will be an alteration of the previous one. The new copy may be darker or lighter than the previous one, and new corruptions may be introduced to the old copy. The OCR interpretation of a fifth-hand copy is likely to contain more errors than the OCR interpretation of the cleaner original copy.

Each legal document was available as a Microsoft Word file and as a text file. The Word versions were printed and fifth-hand copies were made. Each fifth-hand copy was scanned through a machine called the Hodaka that used an OCR technology known as Textbridge. The OCR engine interpreted each image of the fifth-hand copies and created text files of the interpretations. These interpretations were in turn compared to the original text files.
To compare the original text to the OCR of the fifth-hand copy, we needed to write some code that detects differences between the text files. Statistics on the length of runs of mismatched characters were computed as a part of the third model of the error-correction. Before runs of errors were measured, the text files must be preprocessed. Each file is read as a list of single lines of text. The first step is to remove all blank lines from both files. This is to ensure that extra blank lines do not cause different lines of text to be compared against each other.

Then for each line of text, all spaces (including newlines) at the beginning and end of the lines are removed. Extra spaces are deleted so that exactly one space occurs between words. It is not crucial for the OCR engine to get the exact number of spaces between words correct; actual text is what matters. Finally, tabs are converted into spaces so that the code does not make any distinction between tabs and spaces.

Once the text is preprocessed, each pair of corresponding lines is compared letter for letter. When a mismatch occurs, a counter begins to count as long as the letters do not match. When letters finally do match again, the counter stops, but the run length is recorded in another table.

### 5.2 Experiments

The first run of experiments was made with a flawed set of printouts of the legal documents. The printer had been printing out marks of red ink along with the text. As these printouts were scanned, sometimes the OCR engine would try to interpret the red marks. This caused some difficulties in gathering data. Many times a red mark would appear in a vast area of empty space. The OCR engine would insert a line where none existed, thus causing the code to compare a line in the original text to a previous line in the OCR text. Such errors had to be corrected by hand. A more general problem arises when the OCR engine detects stray marks of any kind and then inserts a line. The assist channel has not been designed to handle such misalignment. Another type of vertical misalignment may happen when the OCR engine skips a line entirely. This would happen if the original document contained
a very short line of text such as a single period alone on a line. Such cases are very rare, but they have appeared in our data set.

Because the stray marks from the printer caused much trouble, a new batch of documents were printed. Fifth-hand copies were made for these cleaner documents, and this time there were no extraneous marks to interfere with the OCR process. The original printouts of the clean documents were scanned, and so were the fifth-hand copies.

The results showed significant deterioration in the OCR interpretation of the cleaner documents. The clean batch yielded a character success rate of 99.80 percent (see Table 4.1), while the fifth-hand copies yielded a success rate of 99.16. This implies a fourfold jump in the error rate. The character substitution rates were 0.07 and 0.39 percent, while synchronization errors occurred at a rate of approximately 0.13 and 0.45 percent. Here, a run of at least two mismatched characters is considered to be a synchronization error. See Appendix A for the entire distributions.

After collecting numerical data on the error rates of both batches of documents, data was collected on the frequency of character substitution errors. The code that was written for this task was similar to the code that did the comparison between the text files. Instead of recording the lengths of the runs of errors, the code would record the characters that were swapped in each run of length one. This code could generate a confusion frequency matrix for the data.

Some interesting observations were made from the confusion data. One especially notable observation was that many errors involved square brackets. Perhaps the most common substitution error had the right bracket (]) turn into lowercase l. Other common mistakes had either bracket turning into l, J, t, or f.

The legal documents had an unusually high number of square brackets. They had originally contained blank lines in which the client would place his name, address, the date, etc. Within the lines (which were formed with underlines), a number would be enclosed within square brackets such as [9] (this is not a reference!).

A new batch of documents was created in which most brackets were eliminated. Instead of square brackets, each figure such as [9] was replaced with either "blank9,"
Table 5.1: Required error-correction strength for fifth-hand copies of original documents

<table>
<thead>
<tr>
<th>$k$</th>
<th>$10^{-3}$</th>
<th>$10^{-4}$</th>
<th>$10^{-5}$</th>
<th>$10^{-6}$</th>
<th>$10^{-7}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.7916</td>
<td>3.0731</td>
<td>3.32999</td>
<td>3.56773</td>
<td>3.78999</td>
</tr>
<tr>
<td>2</td>
<td>2.65948</td>
<td>2.93394</td>
<td>3.18311</td>
<td>3.41285</td>
<td>3.62704</td>
</tr>
<tr>
<td>3</td>
<td>2.72905</td>
<td>3.01416</td>
<td>3.27216</td>
<td>3.50948</td>
<td>3.73036</td>
</tr>
<tr>
<td>4</td>
<td>2.84383</td>
<td>3.1434</td>
<td>3.41382</td>
<td>3.66213</td>
<td>3.89295</td>
</tr>
<tr>
<td>5</td>
<td>2.96468</td>
<td>3.27883</td>
<td>3.56184</td>
<td>3.82136</td>
<td>4.06233</td>
</tr>
<tr>
<td>6</td>
<td>3.08796</td>
<td>3.41667</td>
<td>3.71229</td>
<td>3.98304</td>
<td>4.23425</td>
</tr>
<tr>
<td>7</td>
<td>3.20133</td>
<td>3.54322</td>
<td>3.85024</td>
<td>4.13116</td>
<td>4.39159</td>
</tr>
<tr>
<td>8</td>
<td>3.30954</td>
<td>3.66388</td>
<td>3.98167</td>
<td>4.27218</td>
<td>4.54133</td>
</tr>
</tbody>
</table>

or the brackets were eliminated entirely so that only the 9 remains. In effect, we would be filling the blanks with words so that the legal documents represent final documents.

Fifth-hand copies were made for this new batch of documents. The same experiments were run on this new set of copies. The character reading success rate was 99.41 percent. Replacement errors made up another 0.15 percent, and synchronization errors made up the other 0.44 percent. (See Appendix A for the full distribution.)

### 5.3 Optimal Block Sizes

Calculations were made for each batch of documents for what the optimal block size should be. The full calculations are listed in Appendix B. For the original, unaltered legal documents, the optimal size is two columns per block. Tables 4.3 and 4.5 show the effectiveness of each block size. For the corresponding fifth-hand copies, the optimal block size is also two columns per block. Table 5.1 shows the error-correction strength necessary for each block size and page failure rate, for a page of 40 lines and 100 columns.

Finally, for the fifth-hand copies of the documents in which the brackets were replaced, the optimal size is just one column per block. Table 5.2 show the required error-correction strength, once again for a page with 40 rows and 100 columns.

The general tendency is to prefer smaller block sizes as the synchronization errors
Table 5.2: Required error-correction strength for fifth-hand copies of modified documents

outweigh the replacement errors.

5.4 Confusion Frequencies of Textbridge

Next, the replacement errors were analyzed. Each replacement error could be represented as an ordered pair \((a, b)\) where \(a\) is the correct character, and \(b\) is the wrong character. After sorting the errors on the first half of the ordered pair, some interesting observations were made. For instance, there is a significant asymmetry in the confusion matrix. One frequent mistake was the transformation of the comma to a period. Yet not once was a period ever mistaken to be a comma. Other examples of asymmetry included \((s, a), (o, a), (r, i), (r, t), \) and \((f, t)\). The confusion matrix is not totally asymmetric, however. The lowercase letters i and l were frequently switched.

Another notable pattern showed that uppercase and lowercase letters were often confused. Perhaps the errors \((i, I)\) and \((I, i)\) were the most common in this case. Other such mistakes include \((c, C), (C, c), (s, S), \) and \((H, h)\). Thus Textbridge may also be using topological features to determine letters.
Chapter 6

Combining Separation Coding and Error Correction

Up to this point, we have discussed how separation coding and error correction is used in the implementation of an assist channel. Each method has its own strengths and weaknesses. While separation coding reduces the most common errors in OCR, it does not eliminate errors in which characters within the same group are confused. Error correction alone may catch one of these mistakes that separation coding would not catch, but error correction alone fails if the number of errors in a block exceed the correction capability of the code.

In this chapter, we demonstrate how to combine the techniques of separation coding and error correction. In combining these two techniques, we can utilize the strengths of each technique to reduce the OCR error rate even further.

6.1 The Combined Procedure

As before, we first partition the alphabet into groups in which letters that look similar to each other go into different groups. The algorithm for doing the partition is the same as the one in Chapter 3. Each group has a $n$-bit label. The additional feature is that within each group, each character has its own member label. For instance, if there at least 16 but no more than 32 characters in a particular group, each letter in
that group is assigned a five-bit member label. No two characters in the same group may have the same member label, but characters from different groups may share the same member label. So if \( a \) is in group 1 and \( b \) is in group 2, then \( a \) and \( b \) may have the same member label since they are in different groups. In effect, there are two parts to the identification of a character: the group’s label, and the member label within its group.

This combined procedure uses two separate error correction procedures. As before, we divide the page into columns, and each block will consist of the same number of columns. (This time, however, the block sizes need not be the same for both error corrections.) The first error correction is done on the group labels of the letters in the block. The second error correction is done on the member labels of each letter. Thus the assist channel will contain two sets of check digits in which each set corresponds to one of the error-correcting procedures.

The number of groups used for separation coding is not big; typically either 2 or 3 bits will be used for the labeling, and thus there would be a total of 4 or 8 groups. Recall that the length of a block in a Reed-Solomon code is one less than the number of symbols in the alphabet. Since the rows on the page usually outnumber the groups, the group labels alone will not form an alphabet large enough to be used with blocks of size 40 or greater. Thus we form clusters out of the group labels. Let’s say each group label has 2 bits (so there are 4 groups). We might form 6-bit clusters by taking the group labels of three consecutive characters and putting them together. If an error occurs in any one of the group labels, the entire cluster becomes an error.

The blocks in the first error-correction process will usually contain just one cluster per line. (Increasing the number of clusters will not make error correction more effective.) Once we run through the first error correction process, the clusters and group labels should become correct. If a group label is found to be in correct, the OCR engine will make a second guess at the corresponding character, given knowledge of the correct group. Thus the first error correction catches the mistakes that separation coding can fix.

The second error-correction process is done on the member labels of each character.
Any remaining errors that result from substituting characters from within the same group will be detected in this round. The strength of the first error correction should be relatively strong compared to the second one. The error rate of substitutions within a group is far lower than that of intergroup substitutions.

The full details of the combined procedure are still in the making. For instance, desynchronization can affect the proper formation of clusters. If a character is inserted into a cluster by mistake, then the last character of that cluster becomes a part of the next cluster.

6.2 Analysis

In this section, we do a simple back-of-the-envelope analysis of the improvement in OCR that this combined procedure may bring.

Let $p$ be the overall confusion (i.e. single-character replacement) rate of the OCR engine. Let $b$ be the size of a group label, in bits. From the theorem in Chapter 3, a reduction by a factor of $1/2^b$ is possible with separation coding. The error rate for individual group labels is between $(1 - 2^{-b})p$ and $p$. Then the failure rate for recognizing clusters is approximately $sp$, where $s$ is the number of letters in a cluster.

We can derive a formula for the total number of bits $\beta_1$ needed to ensure that the page failure rate $P_1$ of the first error correction is less than some certain threshold. If there are $\beta_1$ bits allocated for the first error correction process, then at most $\beta_1/(bs)$ "check clusters" appear in the assist channel. If a page has $c$ columns, then it has $c/s$ blocks of clusters. Thus at most $\beta_1/(bc)$ check clusters correspond to each block, and no more than $\beta_1/(2bc)$ errors can occur in a block. We let $\mu_1 = nsp$, which is the expected number of cluster failures in a block. We also let $\sigma_1 = \sqrt{nsp(1 - sp)}$ be the standard deviation of cluster failures in a block. Finally, we also need to calculate the maximum possible block failure rate $B_1$. Then using the similar derivation in Section 4.5, we have

$$B_1 = 1 - (1 - P_1)^{s/c}$$
\[
\beta_1 / 2bc = \mu_1 + \sigma_1 Q^{-1}(B_1)
\]
\[
\beta_1 = 2bc\mu_1 + 2bcs_1Q^{-1}(B_1).
\]

If the first error-correction went perfectly, then the reinterpreted document should have no desynchronizations.

We could do a similar calculation for the number of bits \( \beta_2 \) needed so that the page failure rate for the second error correction process is under a threshold \( P_2 \). The intra-group error rate (i.e. replacing characters with another character within the same group) is no higher than \( p/2^b \). We may presume that there are \( 7 - b \) (or perhaps \( 8 - b \)) bits in a member label. If each block has \( k \) columns, then the expected number of errors in a block \( \mu_2 = nk p/2^b \). The standard deviation is \( \sigma_2 = \sqrt{nkp/2^b(1 - p/2^b)} \).

The derivation for \( \beta_2 \) follows:

\[
B_2 = 1 - (1 - P_2)^{(k/e)}
\]
\[
\frac{\beta_2}{2(7 - b)c} = \mu_2 + \sigma_2 Q^{-1}(B_2)
\]
\[
\beta_2 = 2(7 - b)c\mu_2 + 2(7 - b)c\sigma_2 Q^{-1}(B_2).
\]

Further analysis needs to be done on combining separation coding and error correction. Specifically, we would like to determine what proportion of check bits in the assist channel should be used for the first error correction versus the second error correction. (What proportion achieves the lowest page failure rate? How does this page failure rate compare to using just one of the techniques alone?)
Chapter 7

Conclusion and Future Work

7.1 Summary of Completed Work

In this thesis, we have explained the motivation, techniques, and challenges for designing an assist channel to improve optical character recognition. This thesis primarily laid the groundwork for designing an assist channel. We have explored the possible ways of modeling optical character recognition and their error rates. We have demonstrated the accuracies and inaccuracies of these models by experimenting with physical documents.

We have developed algorithms that create an effective separation code that reduces confusions of similar characters. By partitioning the set of characters into \( n \) groups, the reductions are guaranteed to be at least a factor of \( 1/n \). In practice, the reductions are usually much greater.

We also learned how to compute a block size for which error correction can best improve the accuracy of OCR. The error correction should be strong enough to withstand the transformations and corruptions that may occur during repeated photocopying.

7.2 Future Work

Much remains to be explored in the realm of assist channel coding. First, we need to conduct more experiments. Only by collecting more data can we more accurately
capture the nature of how errors are generated during OCR. With additional data, we could get a more accurate estimate of the probability distribution of the lengths of runs of errors.

Studies should also be done on different types of documents so that we can see how models can differ. For example, experiments could also be done with documents of different languages. The separation code for English text could be very different from the separation code developed from, for example, German text. (Not to mention that the German alphabet includes letters such as ä and ö.)

There are additional ideas for dealing with synchronization errors within a line. One is the notion of guessing the next character during the error-correction process. Imagine playing a game in which the player has the OCR’s first interpretation of a line of text. The correct version is slowly revealed, segment by segment. The player must try to guess the next segment to be revealed. The object would be to make as few errors as possible in projecting the next block. If we can work out an algorithm that plays this game effectively, then desynchronizations will minimally disrupt error correction procedures.

Other desynchronization problems involve line skipping and line insertion. These are the same problems encountered during the data gathering process. If the OCR engine skips a very short line by mistake, the character in the following lines will be out of place when error correction is performed. Similarly, the OCR engine might recognize stray marks as characters and insert a line that does not belong in the page.

Finally, future work should be done on implementing an actual assist channel. Now that we have laid the groundwork for designing an assist channel, the next step is to print out pages with assist channels in the margins and see how they actually contribute to improving OCR. Certainly new issues will arise when assist channels are printed and scanned into the OCR engine. Once these issues are resolved, tests can be devised to measure the improvement in OCR that assist channels provide. Assist channels can then be used to carry out the promise of bridging the physical paper world with the digital world.
Appendix A

Distribution Tables
Table A.1: Probability distribution of runs of length $i$ in fifth-hand copies of original documents.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.991633</td>
</tr>
<tr>
<td>1</td>
<td>0.00388868</td>
</tr>
<tr>
<td>2</td>
<td>0.000681902</td>
</tr>
<tr>
<td>3</td>
<td>0.000313306</td>
</tr>
<tr>
<td>4</td>
<td>0.000313306</td>
</tr>
<tr>
<td>5</td>
<td>0.000110579</td>
</tr>
<tr>
<td>6</td>
<td>0.000350166</td>
</tr>
<tr>
<td>7</td>
<td>0.0000737191</td>
</tr>
<tr>
<td>8</td>
<td>0.0000921489</td>
</tr>
<tr>
<td>9</td>
<td>0.0000737191</td>
</tr>
<tr>
<td>10</td>
<td>0.00246959</td>
</tr>
</tbody>
</table>

Table A.2: Probability distribution of runs of length $i$ in fifth-hand copies of modified documents.

<table>
<thead>
<tr>
<th>$i$</th>
<th>$p_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.994109</td>
</tr>
<tr>
<td>1</td>
<td>0.00153787</td>
</tr>
<tr>
<td>2</td>
<td>0.000345888</td>
</tr>
<tr>
<td>3</td>
<td>0.000287353</td>
</tr>
<tr>
<td>4</td>
<td>0.000223497</td>
</tr>
<tr>
<td>5</td>
<td>0.000122391</td>
</tr>
<tr>
<td>6</td>
<td>0.000228818</td>
</tr>
<tr>
<td>7</td>
<td>0.000164962</td>
</tr>
<tr>
<td>8</td>
<td>0.000143677</td>
</tr>
<tr>
<td>9</td>
<td>0.000164962</td>
</tr>
<tr>
<td>10</td>
<td>0.00267132</td>
</tr>
</tbody>
</table>
### Appendix B

#### Tables of Calculations

<table>
<thead>
<tr>
<th>$k$</th>
<th>mean ($\mu$)</th>
<th>second moment</th>
<th>variance</th>
<th>standard dev. ($\sigma$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00836712</td>
<td>0.00836712</td>
<td>0.00829711</td>
<td>0.0910885</td>
</tr>
<tr>
<td>2</td>
<td>0.0211427</td>
<td>0.0300995</td>
<td>0.0296525</td>
<td>0.172199</td>
</tr>
<tr>
<td>3</td>
<td>0.0376404</td>
<td>0.0707304</td>
<td>0.0693136</td>
<td>0.263275</td>
</tr>
<tr>
<td>4</td>
<td>0.0575387</td>
<td>0.135616</td>
<td>0.132306</td>
<td>0.363738</td>
</tr>
<tr>
<td>5</td>
<td>0.0805159</td>
<td>0.228843</td>
<td>0.22236</td>
<td>0.471551</td>
</tr>
<tr>
<td>6</td>
<td>0.106455</td>
<td>0.355477</td>
<td>0.344144</td>
<td>0.586638</td>
</tr>
<tr>
<td>7</td>
<td>0.135</td>
<td>0.517002</td>
<td>0.498777</td>
<td>0.706242</td>
</tr>
<tr>
<td>8</td>
<td>0.166074</td>
<td>0.717695</td>
<td>0.690115</td>
<td>0.830732</td>
</tr>
</tbody>
</table>

Table B.1: Error statistics for different segment lengths (fifth-hand copies of original documents)
### Table B.2: Block failure rates for original set of documents

<table>
<thead>
<tr>
<th>$k$</th>
<th>.5</th>
<th>1</th>
<th>1.5</th>
<th>2</th>
<th>2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.124071</td>
<td>0.00192193</td>
<td>1.85924 · 10^{-6}</td>
<td>9.93424 · 10^{-11}</td>
<td>2.77556 · 10^{-16}</td>
</tr>
<tr>
<td>2</td>
<td>0.1446</td>
<td>0.00188805</td>
<td>1.10781 · 10^{-6}</td>
<td>2.53103 · 10^{-11}</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>0.184732</td>
<td>0.00347559</td>
<td>3.38376 · 10^{-6}</td>
<td>1.46375 · 10^{-10}</td>
<td>2.77556 · 10^{-16}</td>
</tr>
<tr>
<td>4</td>
<td>0.230165</td>
<td>0.00662335</td>
<td>0.000012443</td>
<td>1.30357 · 10^{-9}</td>
<td>7.16094 · 10^{-15}</td>
</tr>
<tr>
<td>5</td>
<td>0.275377</td>
<td>0.0115082</td>
<td>0.000391258</td>
<td>9.20987 · 10^{-9}</td>
<td>1.40943 · 10^{-13}</td>
</tr>
<tr>
<td>6</td>
<td>0.319372</td>
<td>0.0184618</td>
<td>0.00106213</td>
<td>5.16232 · 10^{-8}</td>
<td>1.98747 · 10^{-12}</td>
</tr>
<tr>
<td>7</td>
<td>0.360094</td>
<td>0.0270914</td>
<td>0.00239228</td>
<td>2.09566 · 10^{-7}</td>
<td>1.71443 · 10^{-11}</td>
</tr>
<tr>
<td>8</td>
<td>0.398092</td>
<td>0.0374615</td>
<td>0.0047729</td>
<td>6.95711 · 10^{-7}</td>
<td>1.08446 · 10^{-10}</td>
</tr>
</tbody>
</table>

### Table B.3: Page failure rates for original set of documents

<table>
<thead>
<tr>
<th>$k$</th>
<th>mean ($\mu$)</th>
<th>second moment</th>
<th>variance</th>
<th>standard dev. ($\sigma$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00589074</td>
<td>0.00589074</td>
<td>0.00585604</td>
<td>0.0765248</td>
</tr>
<tr>
<td>2</td>
<td>0.0160997</td>
<td>0.0248054</td>
<td>0.0245462</td>
<td>0.156672</td>
</tr>
<tr>
<td>3</td>
<td>0.0302645</td>
<td>0.0636707</td>
<td>0.0627548</td>
<td>0.250509</td>
</tr>
<tr>
<td>4</td>
<td>0.0480832</td>
<td>0.128415</td>
<td>0.126103</td>
<td>0.35511</td>
</tr>
<tr>
<td>5</td>
<td>0.0693201</td>
<td>0.224379</td>
<td>0.219574</td>
<td>0.468587</td>
</tr>
<tr>
<td>6</td>
<td>0.0938422</td>
<td>0.357111</td>
<td>0.348305</td>
<td>0.590174</td>
</tr>
<tr>
<td>7</td>
<td>0.121411</td>
<td>0.530272</td>
<td>0.515531</td>
<td>0.718005</td>
</tr>
<tr>
<td>8</td>
<td>0.151853</td>
<td>0.747569</td>
<td>0.72451</td>
<td>0.851181</td>
</tr>
</tbody>
</table>

### Table B.4: Error statistics for different segment lengths (fifth-hand copies of edited documents)
### Table B.5: Block failure rates for fifth-hand copies of edited documents

<table>
<thead>
<tr>
<th>$k$</th>
<th>0.5</th>
<th>1</th>
<th>1.5</th>
<th>2</th>
<th>2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0571307</td>
<td>0.000133434</td>
<td>$5.59318 \cdot 10^{-9}$</td>
<td>$3.66374 \cdot 10^{-15}$</td>
<td>0.</td>
</tr>
<tr>
<td>2</td>
<td>0.0855792</td>
<td>0.000353443</td>
<td>$3.23504 \cdot 10^{-8}$</td>
<td>$5.69544 \cdot 10^{-14}$</td>
<td>0.</td>
</tr>
<tr>
<td>3</td>
<td>0.129358</td>
<td>0.00125165</td>
<td>$4.40633 \cdot 10^{-7}$</td>
<td>$4.88115 \cdot 10^{-12}$</td>
<td>0.</td>
</tr>
<tr>
<td>4</td>
<td>0.177575</td>
<td>0.00340847</td>
<td>$3.61725 \cdot 10^{-6}$</td>
<td>$1.8323 \cdot 10^{-10}$</td>
<td>$4.44089 \cdot 10^{-16}$</td>
</tr>
<tr>
<td>5</td>
<td>0.226171</td>
<td>0.00737113</td>
<td>$0.0000184737$</td>
<td>$3.06966 \cdot 10^{-9}$</td>
<td>$3.19189 \cdot 10^{-14}$</td>
</tr>
<tr>
<td>6</td>
<td>0.27365</td>
<td>0.0135778</td>
<td>$0.0000676134$</td>
<td>$2.91067 \cdot 10^{-8}$</td>
<td>$1.02041 \cdot 10^{-12}$</td>
</tr>
<tr>
<td>7</td>
<td>0.318448</td>
<td>0.0220294</td>
<td>$0.000188974$</td>
<td>$1.72986 \cdot 10^{-7}$</td>
<td>$1.58984 \cdot 10^{-11}$</td>
</tr>
<tr>
<td>8</td>
<td>0.360265</td>
<td>0.032605</td>
<td>$0.000434416$</td>
<td>$7.32522 \cdot 10^{-7}$</td>
<td>$1.46904 \cdot 10^{-10}$</td>
</tr>
</tbody>
</table>

### Table B.6: Page failure rates for fifth-hand copies of modified documents

<table>
<thead>
<tr>
<th>$k$</th>
<th>0.5</th>
<th>1</th>
<th>1.5</th>
<th>2</th>
<th>2.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.997213</td>
<td>0.0132556</td>
<td>$5.59318 \cdot 10^{-7}$</td>
<td>$3.66374 \cdot 10^{-13}$</td>
<td>0.</td>
</tr>
<tr>
<td>2</td>
<td>0.988589</td>
<td>0.01752</td>
<td>$1.61752 \cdot 10^{-6}$</td>
<td>$2.84772 \cdot 10^{-12}$</td>
<td>0.</td>
</tr>
<tr>
<td>3</td>
<td>0.990122</td>
<td>0.0408885</td>
<td>$0.0000146877$</td>
<td>$1.62707 \cdot 10^{-10}$</td>
<td>0.</td>
</tr>
<tr>
<td>4</td>
<td>0.992459</td>
<td>0.0818158</td>
<td>$0.0000904273$</td>
<td>$4.58075 \cdot 10^{-9}$</td>
<td>$1.11022 \cdot 10^{-14}$</td>
</tr>
<tr>
<td>5</td>
<td>0.994072</td>
<td>0.137542</td>
<td>$0.00036941$</td>
<td>$6.13933 \cdot 10^{-8}$</td>
<td>$6.39488 \cdot 10^{-13}$</td>
</tr>
<tr>
<td>6</td>
<td>0.99515</td>
<td>0.203753</td>
<td>$0.00112629$</td>
<td>$4.85112 \cdot 10^{-7}$</td>
<td>$1.70068 \cdot 10^{-11}$</td>
</tr>
<tr>
<td>7</td>
<td>0.995818</td>
<td>0.27256</td>
<td>$0.00269625$</td>
<td>$2.47122 \cdot 10^{-6}$</td>
<td>$2.2712 \cdot 10^{-10}$</td>
</tr>
<tr>
<td>8</td>
<td>0.996242</td>
<td>0.339224</td>
<td>$0.00541666$</td>
<td>$9.15649 \cdot 10^{-6}$</td>
<td>$1.83631 \cdot 10^{-9}$</td>
</tr>
</tbody>
</table>
Bibliography


