

APPLICATION OF GENETIC ALGORITHMS TO RECOVERING
CORRUPTED FILE STREAMS

by

John Phillips

A Thesis

Presented to the Faculty of
Bucknell University
In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science with Honors in Computer Science
April 2004

Approved: _____

Stephen Guattery
Thesis Advisor

Gary Haggard
Chair, Department of Computer Science

Acknowledgments

I first would like to thank Professors David Kelley of the Bucknell Electrical Engineering Department, James Lavine of the Bucknell Foreign Languages Department, and Martin Ligare of the Bucknell Physics Department, all of whom provided much needed guidance in helping me research their fields of study. We would not have been able to understand random noise generation, statistical linguistics, or quantum cryptography to the level that we do now, without their much valued help.

I would also like to thank Michael Frey of Bucknell's Mathematics Department both for helping me to brush up on statistical analysis, understand the deflection criterion, and also for sitting on my examination committee.

I would like to thank Jim Van Fleet for helping me find the necessary articles to research quantum cryptography and genetic algorithms.

I would like to thank the network administrators at Bucknell for increasing my computational limits when my program got very large.

I would like to thank Professor Daniel C. Hyde for sitting on my examination committee.

I would like to thank Professor Gary Haggard of Chair of the Bucknell Computer along with all of the other Computer Science professors for their stewardship of the Bucknell Computer Science Department.

Most especially, we would like to thank my advisor Stephen Guattery of the Bucknell Computer Science Department. Without his initial course on Algorithms and his commitment to me and my project we am sure that we would not have succeeded near the level that we have. Specifically, we would like to thank him for helping me debug my thousands of lines of code and for his contributions in many innumerable ways to my project.

Finally, we would like to thank my parents for their continued support.

Table of Contents

| | |
|--|-------------|
| List of Tables | vii |
| List of Figures | viii |
| Abstract | ix |
| 1 Introduction | 1 |
| 1.1 Description of Problem | 1 |
| 1.2 Explanation of Genetic Algorithms | 2 |
| 1.3 Explanation of Quantum Cryptography | 3 |
| 1.4 Overview of Thesis | 3 |
| 2 Text analysis | 6 |
| 2.1 Basic Assumptions | 6 |
| 2.2 The English Language and Written Structuring | 7 |
| 2.3 ASCII | 8 |
| 2.4 Preprocessing Issues | 11 |
| 2.5 Determining the Level of Corruption | 12 |

| | | |
|----------|---|-----------|
| 2.6 | Scrolling Windows and End-of-sentence Profile Matching | 14 |
| 2.7 | Optimization and Deflection Criterion | 17 |
| 3 | Genetic Algorithm | 18 |
| 3.1 | Standard Genetic Algorithms | 18 |
| 3.2 | Questions Moving Forward | 20 |
| 3.3 | Creating and Breeding a Population | 21 |
| 3.4 | Word Matching: The Basis for a Fitness Function | 23 |
| 3.5 | Dictionary and Hash Table | 24 |
| 3.6 | Overview of Genetic Algorithm | 24 |
| 3.8 | Additional Considerations | 26 |
| 3.9 | Performance of the Genetic Algorithm and Ideas for Future | 27 |
| 4 | Application to Quantum Cryptography | 30 |
| 4.1 | Overview of Quantum Cryptography and Terms | 30 |
| 4.2 | Quantum Cryptography vs. Other Types of Cryptography | 33 |
| 4.3 | Mock Protocol | 34 |
| 4.4 | Susceptibility to Corrupted File Recovery | 35 |

| | | |
|----------|--|------------|
| 4.5 | Suggested Protocol Modifications | 38 |
| 5 | Conclusion | 39 |
| 6 | Bibliography | 42 |
| A | Corruption Implementation | 43 |
| B | Text Analysis Implementation | 49 |
| C | Genetic Algorithm Implementation | 60 |
| D | Dictionary/Hash Tables Implementation | 86 |
| E | Example Run of Genetic Algorithm | 102 |

List of Tables

| | | |
|-----------|--|----|
| Table 2.1 | ASCII Representaion (www.asciitable.com) | 10 |
| Table 2.2 | Example of End of Sentence Search | 16 |
| Table 3.1 | Example of Genetic Algorithm Ouputs & Distance from Original | 28 |

List of Figures

| | |
|---|----|
| Figure 2.1 Bernoulli trial formula | 12 |
| Figure 3.1 2-D Representation of Character Combinations | 21 |
| Figure 4.1 Brassard and Bennett's Quantum Encryption [7] | 32 |
| Figure 4.2 Translation scheme: 2-bits encryption key per 1-bit transmission | 35 |
| Figure 4.3 Vertical/Horizontal – Diagonal photon splitting orientation | 36 |

Abstract

The Genetic Algorithms process that attempts to generate near-optimal solutions (answers) to complex problems for which no efficient process of generating optimal solutions exists, provides a promising, yet unexplored, method for reconstructing corrupted texts. This is made possible, mostly because text files use American Standard Code for Information Interchange (or ASCII) as their encoding protocol. ASCII allows for the reconstruction of its texts because, among other reasons, the majority of ASCII characters are rarely used. Furthermore, the reconstruction of corrupted texts allows for an interesting application to decoding some simple protocols of Quantum Cryptography, a process used for transmitting secure data through an encrypted data stream using quantum mechanical properties, thought to be more secure. In this honors thesis we will attempt to provide sufficient evidence that ASCII text processing and Genetic Algorithms, when used in conjunction, can be a promising method for reconstructing corrupted texts. Furthermore, we will attempt to show how some simple Quantum Cryptographical protocols can be reduced to the problem of reconstructing corrupted texts, and therefore, can be broken.

Chapter 1

Introduction

During the spring semester of 2003 I participated in an elective course within the computer science department: “Introduction to Analysis of Algorithms.” The course, taught by Professor Stephen Guattery, allowed me to study both genetic algorithms (a process that attempts to generate near-optimal solutions to complex problems for which no efficient process of generating optimal solutions (answers) exists) and quantum cryptography (a process used for transmitting secure data through an encrypted data stream using quantum mechanical properties).

I enjoyed the material so much that I started to delve into both topics further over the following summer and, when given the opportunity in the Fall, I submitted an honors thesis proposal that would allow me to continue researching both: trying to defeat quantum cryptography by using genetic algorithms.

1.1 Description of Problem

Whenever computer information, or data, is transmitted from one place to another, it is subject to random corruption, and the bits conveying the information may be received as the opposite value (i.e., a one changes to a zero). This leads to various problems in the computer world such as corrupt files, noise in audio channels, etc., which are all very hard effects to reverse and recover the original sequence of bits [5]. These problems are the subject of much research that has produced a number of useful

applications. In particular, error-correcting-codes have been developed to combat the problem of unreliable communication channels. This thesis proposes a new approach to this problem.

1.2 Explanation of Genetic Algorithms

Many problems that computer scientists encounter are very hard to solve. Some of these problems, commonly called NP-hard problems, a subset of which are NP-complete problems, have no known efficient solution process (i.e., no algorithm that returns a solution in a time that is polynomial with respect to the size of the input). An example of such a problem is developing an itinerary for a traveling salesperson where the goal is to visit each city on a predefined list once and only once, while traveling the shortest total distance possible. This problem is commonly referred to as the TRAVELING SALESMAN PROBLEM (TSP). TSP is an optimization problem. While any ordering of the cities is a possible solution, there will only be a few solutions that have the shortest distance possible. The question, then, is not whether a solution to the problem exists, but what is the optimal solution to the problem.

While no efficient process is known for generating an optimal solution all of the time to an NP-hard problem, many NP-hard problems can be solved efficiently much of the time to near optimality using a heuristic. A heuristic is a solution-generating rule that gives near-optimal solutions a high percentage of the time. However, there is no guarantee that a heuristic will ever give a near-optimal solution at all. Genetic algorithms are heuristic algorithms that use the idea of genetic recombination and mutation to produce ‘populations’ of possible solutions to a problem that ‘mate’ with each other to produce new (and over a sequence of generations) better possible solutions. By defining

the characteristics of what a good answer should look like (i.e. shorter travel path for TSP) genetic algorithms can use a ‘fitness test’ for the ‘population’ of answers and effectively kill off genetically less fit solutions. Over time, the genetic algorithm will produce a few exceptionally fit individual solutions within the population. These individual solutions would be highly, but not completely, optimized.

1.3 Explanation of Quantum Cryptography

My thesis examines the application of a genetic algorithm capable of reconstructing corrupted character files, to the problem of deciphering a quantum encryption transmission. Quantum cryptography relies on the exploitation of a quantum mechanical phenomenon such as the uncertainty principle or the violation of a Bell inequality to encode data in a way that is statistically very secure from eavesdroppers, or unwanted listeners, on the channel [1]. Quantum encryption uses light photons. Light photons, like other traveling particles or waves, can be observed to determine their properties. The wavelength, amplitude, and intensity of a photon of light can all be measured without changing the photon. However, it is theoretically impossible to measure the polarization of a light photon without the possibility of changing the polarization of the photon, unless the measurer knows exactly what polarization the photon already has.

1.4 Overview of Thesis

My thesis presents research on using genetic algorithms to recover corrupted texts, and furthermore to reduce breaking a quantum cryptographic protocol down to a corrupted text recovery. Initial search of the literature revealed no prior research

conjoining the two topics in this way and so we are led to believe that, barring an unfound research document surfacing, this is a new line of research.

Moreover, my thesis shows that genetic algorithms have a very high potential in being able to reconstruct text corrupted up to 15% corruption. We have also found a way to reduce a Quantum Cryptography protocol to (on average) a 15% corrupted text, in effect breaking that protocol.

This thesis is divided into three main parts. Each part reflects a different stage of processing that an input would undergo if put through all three of our algorithms. In a computer scientist's terms, all of the three middle chapters, Chapters 2, 3 and 4, can be thought of as describing a different module of the process, each having a separate input and an output that feeds into the next.

The thesis is organized as follows; Chapter 1 has provided a brief introduction to the thesis and the description of the problem. The first part of the algorithm, described in Chapter 2, can be thought of as a corrupted file divider. As input it takes a text file that has an unknown $X\%$ of its bits randomly corrupted, and as output it returns two items: First, it returns the file, broken into sentences. It also returns an estimate of the corruption value of the file. However, the characters of the first output are slightly different than those of the entered input. In the process of estimating the corruption level, approximately 12.5% of the corrupted bits are corrected. This part of the algorithm performs better at lower corruption percentages and becomes more imprecise in breaking apart the sentences at higher levels of corruption.

Chapter 3 describes a corrupted sentence reconstructor algorithm. As input this algorithm takes a corrupt sentence and an estimated corruption level. As output it returns

a high-fitness-value specimen from the genetic algorithm that has most, if not all, words from the original sentence correctly identified. This part of the algorithm is also better at lower corruption percentages and becomes imprecise at higher ones.

Chapter 4 describes the reduction of a quantum encryption channel to a corrupt text problem. An eavesdropper on the proposed quantum encryption channel can recover bits from the plain text with a probability of 50%, corresponding to reading a file with a 50% corruption level. However, after the reduction, the same problem is shown to have an overall average corruptness level of 15% if defined in terms of a corrupted text problem.

Should the main parts of the thesis be applied in series to an input, starting with part 3, moving on to part 1, and ending with part 2, the corruption level would decrease from 50% to 15% to about 7-8%

Finally, Chapter 5 concludes this study with a summary of our results and some ideas for future work.

Chapter 2

Text analysis

The first part of the thesis can be thought of as a corrupted file divider. As input it takes a text file that has $X\%$ of its bits randomly corrupted, and as output it returns two items: the input file broken down into sentences, and an estimate of the corruption of the file. The characters of this output are slightly different than those of the entered input. On average each character of the output file is 12.5% less corrupted than the input character because some specific bits are corrected during corruption estimation. This part of the algorithm performs better at lower corruption percentages and becomes more imprecise at higher ones.

This Chapter is divided into seven sections. In section 2.1 we discuss the base assumptions for the problem. In sections 2.2 and 2.3 we discuss the background of the problem and discuss some key assumptions about the nature of the input. In sections 2.4, we discuss the questions we had to answer in order to make the algorithm. And sections 2.5, 2.6, and 2.7, we discuss implementation and optimizing of the algorithm.

2.1 Basic Assumptions

To condense the problem into a workable area some assumptions were made while addressing the problem. First, we had to specify what corruption is, and second what parameters define the input values.

By “corrupted” we mean that if an original bit in a file was a zero then the corrupted bit is a one, and if an original bit was a one then the corrupted bit is a zero. By “reconstruct” we mean that a corrupted bit is returned to its original orientation. By 15% corruption we mean that a file N characters long with $N*8$ bits will have, on average, $N*8*0.15$ bits switch from their original orientation (zero or one) to the alternate orientation.

In referencing how corrupted one specimen is from another we refer to the *bit-wise distance*. That is the number of bits that are different from one specimen to another. Ex: 01110010 has a bit-wise distance of 3 from 01010111.

We also mad the following assumptions about inputs that had been corrupted:

- All inputs are written in the English Language.
- All sentences must end with a ‘period’ and, except for the first sentence, they must also begin with ‘space’, ‘space’ and a ‘capital letter’. The first sentence must begin simply with a ‘capital letter’.
- All inputs have no tabs, commas, or other non-sentence-ending-period punctuation or newline characters.
- All inputs are encoded in ASCII.
- All inputs are spelled correctly.

2.2 The English Language and Written Structuring

While the English language is not the only language that is transferred and stored using computers, it is used by a large portion of the computing community in order to

communicate. From an analysis standpoint this common usage of the English language means that any patterns existing within it can be potentially exploited in order to reconstruct corrupted texts. Luckily, there are many patterns inherent in the English Language. Some patterns in the English language that occur are [3]:

- text-to-space ratios
- average sentence length
- letter frequency: (The most frequent letters, in order, are e-t-a-o-n-i-s-r-h-l-d-c-u [3], [7])
- word frequency: (Some of the more frequent include: the-of-to-in-and-a-for-was [3])
- letter patterns: (i.e. 'qu', certain letters can only follow some letters)
- word composition: common syllables, prefixes, and suffixes

In this thesis, the focus is exclusively on the English language.

Sentence length, word length, sentence arrangement and vocabulary all differ among individual sentences. While English sentences can be grammatically incorrect as well as spelled incorrectly, allowing such variation unnecessarily complicates the initial investigation of the problem. Thus we chose to restrict the sentences to be correctly spelled and constructed sentences. It would be very interesting to extend this research to sentences with improper grammar and incorrect spelling, in the future.¹

2.3 ASCII

¹ While originally part of the plan for generating high genetic algorithm fitness, the system implemented does not consider grammar in its operation. Hence only the assumption about correct spelling applies.

Computer data is stored in many formats. Text files and the base structure for most document formats are based on a character encoding scheme called ASCII, which interprets every eight bits of a file as a printable or nonprintable character. An example of a nonprintable character would be any control character.

ASCII (the American Standard Code for Information Interchange) is the primary, most commonly used character encoding in the United States today. It allows computers to interpret 8-bit numbers as characters, which include letters along with a number of other symbols and keyboard strokes. ASCII was first created in the 1960's and was standardized in 1968. It is the United States Government standard and has been declared such by the American National Standards Institute. Originally, ASCII was a seven bit coding system allowing for 128 distinct characters, but has since been expanded to eight bits or a single byte, representing 256 distinct characters.

Because ASCII is the most used character standard today, we have assumed that test inputs are in ASCII format. This is primarily because most text documents sent over an encryption channel or corrupted in every day use would be encoded using the ASCII standard. However, many other standards are used; most of them are similar to ASCII in their structure. A future extension of this research would be to consider other text encoding schemes.

Working with ASCII is beneficial in attempting to reconstruct corrupted text files. As explained below, the inherent properties of ASCII provide information that is useful in recovering a corrupted file.

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|-----|----|-----|-----------------------------|-----|----|-----|-------|-------|-----|----|-----|-------|-----|-----|----|-----|--------|-----|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 | | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ^ |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | \$ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | (| (| 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 |) |) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [| [| 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 |] |] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | | DEL |

Source: www.asciitable.com

| | | | | | | | | | | | | | | | |
|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|
| 128 | Ç | 144 | É | 161 | í | 177 | ⌘ | 193 | ⊥ | 209 | ⌘ | 225 | β | 241 | ± |
| 129 | ù | 145 | æ | 162 | ó | 178 | ⌘ | 194 | ⌘ | 210 | ⌘ | 226 | Γ | 242 | ≥ |
| 130 | é | 146 | Æ | 163 | ú | 179 | | 195 | ⌘ | 211 | ⌘ | 227 | π | 243 | ≤ |
| 131 | â | 147 | ô | 164 | ñ | 180 | ⌘ | 196 | — | 212 | ⌘ | 228 | Σ | 244 | ∫ |
| 132 | ä | 148 | ö | 165 | Ñ | 181 | ⌘ | 197 | + | 213 | ⌘ | 229 | σ | 245 | ∫ |
| 133 | à | 149 | ò | 166 | ª | 182 | ⌘ | 198 | ⌘ | 214 | ⌘ | 230 | μ | 246 | ÷ |
| 134 | â | 150 | û | 167 | º | 183 | ⌘ | 199 | ⌘ | 215 | ⌘ | 231 | τ | 247 | ≈ |
| 135 | ç | 151 | ù | 168 | ¿ | 184 | ⌘ | 200 | ⌘ | 216 | ⌘ | 232 | ϕ | 248 | ° |
| 136 | ê | 152 | — | 169 | — | 185 | ⌘ | 201 | ⌘ | 217 | ⌘ | 233 | ⊙ | 249 | · |
| 137 | ë | 153 | Ö | 170 | ¬ | 186 | ⌘ | 202 | ⌘ | 218 | ⌘ | 234 | Ω | 250 | · |
| 138 | è | 154 | Û | 171 | ½ | 187 | ⌘ | 203 | ⌘ | 219 | ■ | 235 | δ | 251 | √ |
| 139 | ï | 156 | £ | 172 | ¼ | 188 | ⌘ | 204 | ⌘ | 220 | ■ | 236 | ∞ | 252 | — |
| 140 | î | 157 | ¥ | 173 | ¡ | 189 | ⌘ | 205 | = | 221 | ■ | 237 | φ | 253 | z |
| 141 | ï | 158 | — | 174 | « | 190 | ⌘ | 206 | ⌘ | 222 | ■ | 238 | e | 254 | ■ |
| 142 | Ä | 159 | f | 175 | » | 191 | ⌘ | 207 | ⌘ | 223 | ■ | 239 | ∩ | 255 | |
| 143 | Å | 160 | á | 176 | ⌘ | 192 | L | 208 | ⌘ | 224 | α | 240 | ≡ | | |

Source: www.asciitable.com

Table 2.1 ASCII representation (www.asciitable.com)

ASCII is highly structured. It groups relevant bits very close to one another. As the charts above show, the majority of the bit combinations allowed by ASCII do not encode into characters used in text files. Instead both the of the first thirty-two bit combinations (those of 0 to 31) and of the entire upper 128 (those of 128 to 255) are letters not commonly used, punctuation, digits, or commenting marks. Additionally noteworthy are two other points: First, lowercase (97 to 122) and uppercase (65 to 90) letters differ only by one bit, essentially making a corruption from one to the other easily reversible. Second, the space character, by far the most frequent character in most files, is far, bit-distance wise, from frequently used characters such as letters. None of the values that are only one bit removed from a space ('32') are used frequently when compared with the lowercase letters that make up most words: 'á' (160), '' (96), NULL (0), '0' (48), '(' (40), '\$' (36), "" (34), '!' (33).

2.4 Preprocessing Issues

Preprocessing is important in making the genetic algorithm's job easier. The main questions in deciding how to preprocess corrupt files for input to the genetic algorithm were as follows:

1. How can we determine the corruption level of an input file?
2. Can the corruption level be reduced during preprocessing? If so, then how?
3. What size of text chunk should be sent to the genetic algorithm?
4. How should the file be partitioned into that size chunks?

2.5 Determining the Level of Corruption

Our model of corruption assumes each bit is randomly corrupted with a fixed probability. There is always a chance that the original character will come through uncorrupted in the corrupted file. The statistical model for handling these cases, and in fact any probability case regarding random independent events, is a sequence of Bernoulli trials:

$$f(x) = P[X = x] = \binom{n}{x} p^x (1-p)^{n-x}$$

n = a fixed number of Bernoulli trials
p = the probability of success in each trial
X = the number of successes in n trials

Figure 2.1 Bernoulli trial formula

Estimating the corruption level requires finding something that happens with regular frequency, which can be used as a gauge. The frequency of uncorrupted occurrences of the gauge in the corrupted file, and of (estimated) occurrences of the gauge with one bit corrupted serve as two percentages which could be charted in relation to a universal average, and the Bernoulli probability extrapolated from those two measurements. The extrapolated probability would then become the estimate for the file's corruption.

Initial efforts considered spaces as a possible gauge character. We initially considered analyzing the number of spaces, and close-to-space characters (one bit

difference with a space) in a file, and then comparing that number with the statistical average of how many spaces should be in a file of that length (something that could be derived from the average number of words in a sentence and letters in a word). The further the actual number was from the average number, the more corrupted the file must be.

The problem with this approach was that it depends on the length of the file and the number of spaces in the file. Both of these values could vary wildly from any constant average ratio used in the estimation.

The solution to this problem eventually came in the form of another procedure already being applied to the top order bits of each character. Because ASCII was set up originally with 7 bits, all of the keyboard characters are located in the lower 128 possible bit combinations encoded with values between '00000000' to '01111111'. This means that no characters from the original uncorrupted input file could possibly have its high-order bit set to 1. Any such higher order bit can be considered corrupt and safely set to 0. This provides a quick way to reduce the overall file's corruption level by 12.5% (or $1/8^{\text{th}}$). On average, $1/8^{\text{th}}$ of the bits that were corrupted in each file would occur in this top order bit. This observation also provides a solution to getting an estimate for the corruption level: In addition to switching the top order bits back from '1' to '0', we count the number of times the high-order bit was equal to one in the entire file and divide by the number of characters. This provides a much sharper gauge for measuring corruption than estimating the number of spaces that should occur.

It is still possible that this method could not work for a specific input. There could be a file that is 87.5% corrupted, but has no corruption in the top order bits. In this case, this check would yield a 0.0% corruption score. However, this is a statistical long shot, an estimate of 0.0% corruption would occur far less frequently than the outliers that would cause a breakdown of the original estimation idea based on spaces.

With the top order bit reversed, the file that was, on average, 12.5% less corrupt than the original. To get an estimated corruption value of the file with corrected high order bits the original corruption estimate can be multiplied by .875 (7/8).

2.6 Scrolling windows and end-of-sentence profile matching

Another goal of preprocessing is to break the input into chunks that a genetic algorithm can handle. Three basic units suggest themselves: the entire file, sentences, and single words. A genetic algorithm populated with specimens the size of the entire file would clearly use too many computer resources, and would likely run too slowly to be effective.

On the other end of the spectrum, single words would be too hard and imprecise to identify, the only way of identifying a word being a search for the space characters and close-to-space characters around the word. This might not be a problem in low corruption level files, but as the corruption level rose the spaces would become less distinguishable from other characters. None of the close-to-space characters to which spaces would be corrupted at low corruption rates are commonly used characters. But as the corruption level rises, spaces could frequently end up with two or more corrupted bits,

making them hard to identify. Moreover, in a large text it becomes more likely that a single word could have a corruption value that fluctuated far away from the estimated average of the file, bringing back the fear of a statistical outlier problem.

The third alternative is sentences. This is a good alternative to the other two choices for two main reasons: First, the end of sentences would be easier to pick out of a corrupted file than a space at the end of every word. Because sentences would be typed and be back-to-back, we have assumed all sentence ending/beginning combinations would have a standard format. That is, all sentences would end with a punctuation mark, and if another sentence followed, then the next three characters after the punctuation would be a 'space', 'space', and a 'capital letter', the capital letter starting the next sentence. Second, sentences are a reasonable size that does not require too many resources in the genetic algorithm.

Given that the input would be broken into sentences, we needed to make a search of some sort that would look through the file for a (possibly corrupted) profile matching the pattern: 'punctuation', 'space', 'space', 'capital letter'. Because of time constraints we were forced to look only at the case where all punctuation is periods.

The method for finding the end/beginning of a sentence is straightforward. We knew what a period, a space and a capital letter should all look like bitwise because they all had standard ASCII inputs. The period and the spaces all had an exact 8-bit profile, and all capital letters had the same three top order bits.

The program for separating sentences looks through the entire file, looking four characters at a time, and measures the bit-distance (number of bits different) of the characters in the window from a period, a space or a capital letter. Weights are assigned to the distances for each character. These weights were set initially to 6, 8, 8, 5, and 3 for the period, space, space, top order capital letter bits, and lower order capital letter bits respectively, with the idea that statistic analysis could be done to optimize them later. However, these initial weightings gave very distinct distances between four character entries that were the end of sentences and four character entries that weren't. The values were so good that it was easy to set a threshold separating virtually all end-of-sentence sequences and statistical optimization was not done.²

ORIGINAL FILE:

This is the test file for my honors thesis. This sentence will be my all around test sentence. This is the last sentence.

CORRUPTED TO APPROX 15% CORRUPTION (ACTUAL CORRUPTION IS 0.169355):

```
vXhr`lw rHd0XmgplvXjA foBXmy!XoooVX xXeaqwX
4T(iXdwe9tanwe`gXXlXXa ey0`llXasmu,n06drXXsen$-
."m. !X`yXXXq0vhX`)hX5X3%X|L\XX~X
```

AFTER PREPROCESSING (TOP ORDER BIT SWITCHED BACK):

```
vhhr`lw rHd0|mgplvijA foB'my!hoooVa xheaqw.
4T(icdwe9tanwe`giXlhba ey0`llXasmu,n06drvpsen$-
."m. !X`ysX)q0vh` )hR5$3%l|L\#!~X
```

SCROLLING WINDOW SEARCHING FOR END OF SENTENCES:

```
vhhr`lw rHd0|mgplvijA foB'my!hoooVa xheaqw. Found __C
4T(icdwe9tanwe`giXlhba ey0`llXasmu,n06drvpsen$-."m.
```

Found __C

```
!X`ysX)q0vhX`)hR5$3%l|L\#
```

We assume that the end of the file also ends with a sentence but because the sliding window doesn't detect a space, space, capital letter after the last character it does not output the "Found __C" The following is one example done to 15% corruption: (an X denotes a character that was unprintable; a "__C" denotes the end of a sentence.)

Table 2.2 Example of End of Sentence Search

² Note that the program only works to a high probability. There is no guarantee that it will return the end of every sentence.

Note that, depending on the level of corruption of the file, the weightings of the breakpoint values may need to change. This makes sense because the more corrupt a sentence is, the more likely it is that characters that are not an end-of-sentence will look like an end-of-sentence.

2.7 Optimization and Deflection Criterion

While there was not time to optimize the weighting coefficients involved in the in the sentence breaking algorithm, we note that it can be done using a statistical tool called a deflection criterion.

The analysis will involve computing the average values of all positive end-of-sentence matches of each weight as well as the average values for all non-end-of-sentences and computing the maximum of that difference of the two over the variance of the statistic that the weights define.³

³ Thanks to Professor Michael Frey for explaining the deflection criterion to us and for his overall help in the optimization area.

Chapter 3

Genetic Algorithm

The second part of the thesis, described in this chapter, can be thought of as a corrupted sentence reconstructor. As input, it takes a corrupt sentence and an estimated corruption level (the outputs of the algorithm described in Chapter 2). Experiments have shown that the genetic algorithm output good results at 15% corruption, but we expect to see poorer returns at a higher corruption level. However, our surmise is that this part of the algorithm is also better at lower corruption percentages and becomes imprecise at higher ones.

This chapter is divided into eight sections. In sections 3.1 and 3.2 we discuss the background of the problem and lay out some key assumptions about the nature of the input. In sections 3.3, we discuss the questions we had to answer in order to develop the algorithm. And in sections 3.4, 3.5, 3.6 and 3.7, we discuss the building of the algorithm. In section 3.8, we offer possible areas for optimizing the algorithm along with an example of our results.

3.1 Standard Genetic Algorithms

Genetic algorithms are heuristic algorithms that use the idea of genetic recombination and mutation to produce ‘populations’ of solutions to a problem that ‘mate’ with each other to produce new (and over a sequence of generations) better

solutions. Many problems that computer scientists encounter are very hard to solve. Some of these problems, commonly called NP-hard problems, a subset of which are NP-Complete problems, have no known efficient solution process (i.e., no algorithm that returns a solution in a time that is polynomial with respect to the size of the input). While no efficient process exists for generating an optimal solution all of the time to an NP-hard problem, many NP-hard problems can be solved efficiently much of the time to near optimality using a heuristic. A heuristic is a solution-generating process that gives near optimal solutions a high percentage of the time. However, there is no guarantee that a heuristic will ever give a near-optimal solution at all.

The grand idea behind genetic algorithms is the survival of the fittest, essentially evolution. In real life, a species of animal or plant, etc. will begin to die off over time if a more dominant species is competing with it for the same resources. The weakest specimens of the species start to die off first, while the stronger ones survive and continue to breed. Over time the strong mate with the strong and produce even stronger, or fitter, specimens, and the species adapts to the challenges by evolving a population better able to compete. Genetic algorithms try to mimic this action of nature. They take solutions to problems and, by controlling the death and breeding of the specimens, influence the population to become collectively and individually fitter [6].

Every genetic algorithm has three basic elements. There must first be a group of solutions to a problem, called the population. Secondly, there must be a way of measuring solutions against each other and determining which solution is best, or most fit. This is called a fitness test or fitness function. And lastly, there must be a way of

combining together different solutions from any one population to produce new solutions, called breeding or mating. By continuously breeding the fit specimens of each population with each other the hope is to produce even more fit specimens [6].

The most important aspect of any genetic algorithm then becomes the fitness function that defines what makes a fit solution and what makes for an unfit solution [6].

It should also be noted that many genetic algorithms imitate nature by introducing mutations into child populations, small discrepancies that were not present in either of the parent solutions, hoping, as with nature, to produce a higher fitness in a specimen merely by chance [6].

In order to specify the fitness function for the problem of recovering texts, it is necessary to understand the domain and its definition.

3.2 Questions Moving Forward with the Algorithm

The main questions to be answered in writing the genetic algorithm can be summarized as follows:

1. How would we make our initial population?
2. How would we breed the populations of solutions?
3. What should the data structure of the genetic algorithm look like?
4. What should we give high fitness values to in our fitness function?
5. What type of other abilities would we give our genetic algorithm? and,
6. How could we get the best optimization of our genetic algorithm?

3.3 Creating and Breeding a Population

Because the genetic algorithm receives a single sentence from the sentence breaking function, that specimen serves as the solitary parent to the first generation of specimens. The genetic algorithm also receives an estimate of the level of corruption of the file, and by extension, of the sentence. To produce a population, the user is allowed to define the size of the algorithm's population. Then, each member of the population is created by making a corrupted version of the initial specification, subjecting each bit of the input to the estimated level of corruption.

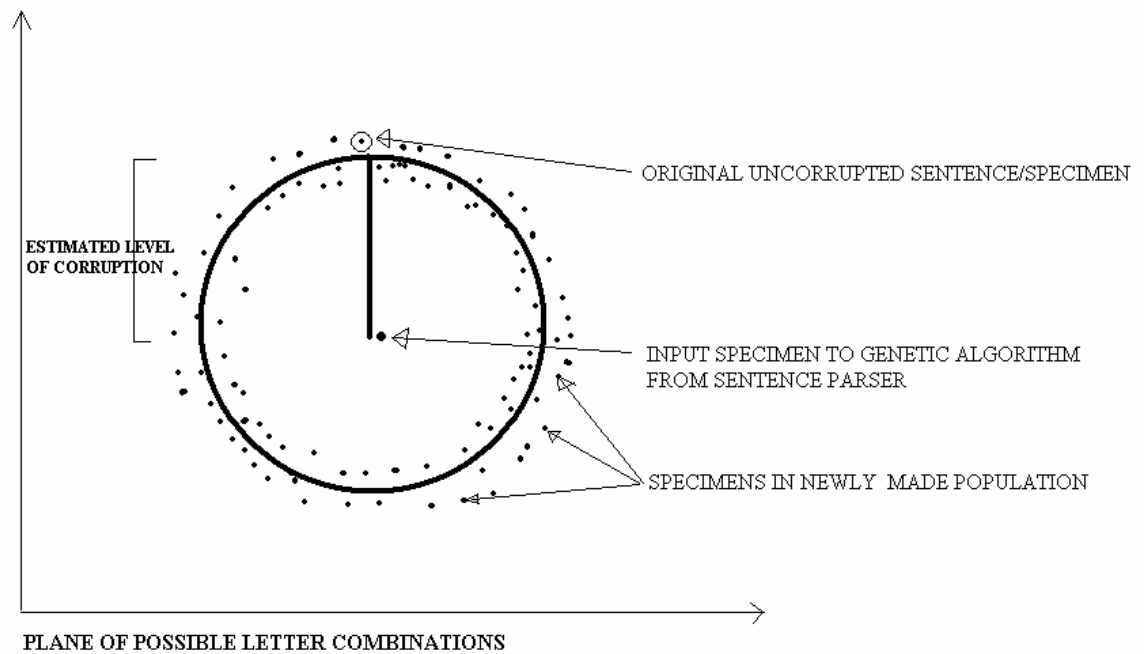


Figure 3.1 2-D Representation of Character Combinations

While many of the new corruptions would take many specimens of the new population even further away from the original uncorrupted value, a few would inevitably be

positioned very close to the original uncorrupted value (see fig 3.1). The fitness function would then be responsible for sorting the closer ones from the farther ones.

Because the problem deals with sentences made up of characters, we decided to breed specimens together by randomly selecting a point and cutting two specimens at that point and switching all the characters that followed that point. This is in contrast to randomly selecting a bit, which could lie in the middle of a character and switching all bits that follow that point.

Given the choices above a data structure should house the population and facilitate the breeding of the population. Moreover, it had to be robust enough to allow for changes later in the genetic algorithm without requiring the entire data structure be revamped. Thus each specimen was given four different arrays to contain four different sets of data particular to each specimen:

1. A character array to hold the specimen's version of the sentence.
2. An integer array of the same length as the first array. This would hold the bit-wise distance that each character was from the original input specimen.
3. An array of floating point values of the same length, to hold the fitness value for each character in the specimen. And finally,
4. Another float array of eight times the length of the previous arrays. This would hold a mutation statistic for each bit in the array. (This information was not used in the algorithm described in this paper but was put in for future use.)

Along with the four arrays, the genetic algorithm has to have efficient access to two statistics for each specimen: the overall fitness value and the corruption level (bit-wise difference from the inputted sentence to the genetic algorithm). Along with the four arrays, these six data members made up the data structure for our genetic algorithm.

3.4 Word Matching: The Basis for a Fitness Function

Our initial idea for the fitness function was to assign a high fitness value to any sentence that included actual words, the more words the higher the fitness score, and very high fitness for any sentence that was grammatically correct; our assumption being that, in order to be grammatically correct a sentence must first contain all real words.

Unfortunately, an open source grammar checker was not available, so we had to remove that from the scope of the fitness function.

Characters were given a separate fitness scores depending on whether or not they were contained within a word, and the total sentence's fitness score was determined by summing up fitness score of each character in the sentence. This, of course, depended upon the fitness function being able to quickly determine if something was or was not a word.

To search out words in each specimen we briefly considered using a scrolling window, much like the one used in the sentence parser of Chapter 2. However, we decided that this method would take too much time per specimen as we would have to have multiple windows of different sizes for words of different sizes.

Instead, we focused on specimen properties. A group of characters was checked to see if it was a word if the group of characters had a space on both sides. This way the exact length of a potential word was known, and it could be accessed immediately. If a group of characters was a word, then the letters in the word along with the spaces surrounding the word would have their individual fitnesses increased in the fitness array. To aid the fitness function, a small fitness value was assigned to any space that did not have a word on either side of it. This way the initial generations of the population would breed together; those sentences would have spaces within them and those spaces would propagate until the middle sections between the spaces would eventually become words.

3.5 Dictionary and Hash Table

To accomplish the goal testing if a string was a word, we first found the largest free dictionary/word bank available on the internet. It was broken up into 22 different files, and each file was used to create a hash table for a different word length [2].

To test if a group of characters was a word, the hash table for the appropriate word length was checked to see if the group of characters was in it.

3.6 Overview of the Genetic Algorithm

The genetic algorithm implemented for this thesis performs the operations described above in the following order (the class and method names refer to code provided in Appendix C; a graphical representation of the data structure can also be found there):

1. The user declares a **GenAlg** object: ortext (a data member holding the input (original) text) array defined.
 - 1.1. The **GenAlg** object calls **Randomize**: using ortext along with its estimated corruption value X, a new population, sized by the user as 'populationsize', is made by flipping bits using a random corruption rate specified in the 'target' array. Target currently is set to the percentage X for all bits other than top order ones.
 - 1.2. For each specimen in Population:
 - 1.2.1. **Randomize** calls **Distance**: Measures the bitwise distance for each character from ortext. Computes the corruption level as measured from ortext by summing across the **Distance** array.
 - 1.2.2. **Randomize** calls **TargetCH**: Computes mutation targets by increasing mutation probability of each character in array if more mutation is desired.
 - 1.2.3. **Randomize** calls **Fitness**: Computes Fitness of each character based on fitness function. Sums array to compute fitnessvalue for each specimen.
 - 1.2.4. **Randomize** calls **Order**: Sorts the population based on fitnessvalues of specimens.
 - 1.3. **GenAlg** calls **Breed** for user input 'generations' iterations.
 - 1.3.1. For each iteration of **Breed** and for each specimen pair in population, **Breed** randomly selects pivot point for neighboring specimens and swaps tails

1.3.1.1.**Breed** calls **Mutate** on both specimens in breeding pair:

Randomly selects one position in specimen. Generates mask based upon TargetCH array. Applies mask to selected character. If the result is not a printable character the character reverts to pre-mask state.

1.3.1.2.**Breed** calls **Distance** on both specimens in breeding pair:

Distance works as described above 1.2.1.

1.3.1.3.**Breed** calls **TargetCH** on both specimens in breeding pair:

TargetCH works as described above 1.2.2.

1.3.1.4.**Breed** calls **Fitness** on both specimens in breeding pair:

Fitness works as described above 1.2.3.

1.3.1.5.**Breed** calls **Order**: First orders new child population based upon fitness, then creates a new population based upon parent population and child population without causing elitism. It does this by choosing the higher fitness specimen from either population making sure that only one specimen of any particular fitness value is selected from the parent population (to avoid elitism).

3.8 Additional Considerations

With the basis of the genetic algorithm set, we considered adding functions and breeding practices that would refine the genetic algorithm and improve its performance.

One breeding practice experimented with was two different forms of elitism. Elitism is a Genetic Algorithms practice where the best of the specimens from the previous generation are allowed to exist through to the next generation. Initially, we took this practice to an extreme, although unintentionally, by taking the top fitness specimens, regardless of whether the specimen had come from the child or the parent population. A consequence of doing this was to fill up the population with a significant number of copies of the highest fitness-valued specimen was, especially if it remained the highest valued specimen for a few generations. This would cause a quick doubling of the top specimen and over a few generations the purging of all other genetic solution material. This negative consequence was immediately rectified once discovered, and the performance of the genetic algorithm improved substantially.

We also made slight changes with respect to mutation function. Instead of mutating a given character in any direction, we only allowed mutations that would take the character to another character. This was done in the initial creation of the population and in subsequent individual mutations.

To optimize the fitness function, the weights assigned to actual words were adjusted. Initially, every space received half a point, and one point was assigned to every character of a word. This worked in getting smaller words, but words larger than 3 or 4 characters eluded the genetic algorithm. To counter this, the number of points assigned to an actual word of more than 3 characters was increased; the longer the word, the more points that word's letters get in the fitness function.

We also observed that the genetic algorithm would start out on a good track but would soon stray too far away from the estimated corruption level to be anywhere close to the original uncorrupted sentence. In order to balance this, a multiple of the absolute value of the file's estimated corruption and the corruption level of the current specimen to the input specimen was subtracted from the fitness of the current specimen. This allowed the genetic algorithm to concentrate on specimens that were of an approximate distance from input specimen as the estimate of the input specimen's initial corruption.

3.9 Performance of the Genetic Algorithm and Ideas for Future

```

Output:  hall bay name ha not a Dr pi nary worn
Bitwise diff: 01005220000000120000000133105000000002 28
Original:hello my name is not a dictionary word 304 9.2%

Output:  helms am name ha nor a Dec a nary word
Bitwise diff: 00013021000000120002000120315000000000 24
Original:hello my name is not a dictionary word 304 7.9%

Output:  haless as name ha not a day a nary word
Bitwise diff: 01013021000000120000000012315000000000 23
Original:hello my name is not a dictionary word 304 7.6%

```

Table 3.1 Example of Genetic Algorithm Outputs & Distance from Original

Note in figure 3.1 the genetic algorithm is able to reduce from around 12.5% corruption to around 7.5%, where a significant portion of the corrupting bits remaining in each specimen come from false positive spaces characters, and account for roughly 33% of corrupted bits.

The genetic algorithm does very well in using each character to produce specimens with all words. However, it has a hard time making larger words and this in

turn causes mediocre corruption level score in final specimens due to them containing false positive spaces.

There is significant opportunity to improve upon the genetic algorithm. One area for future research would be the incorporation of context-specific information into the fitness function. In developing a specific fitness function for specific users, a genetic algorithm could be tailored to individuals and quite possibly made better.

Also, future genetic algorithms could incorporate grammar checking into its fitness function. In doing this the fitness function could change from word to word in the same way that characters currently change. This however, would be dependent upon having grammar checking software, which at this point in time, and is not freely available. Also, future genetic algorithms could look at using common word pairings and syllables. And, finally one future genetic algorithm might look at incorrectly spelled words.

Chapter 4

Application to Quantum Cryptography

The third part of this thesis is a reduction of a quantum encryption channel to a corrupt text problem. The quantum encryption channel starts at an overall average corruptness level of 50%. However, after the reduction the same problem is shown to have an overall average corruptness level of 15% if defined in terms of a corrupted text problem.

This chapter will cover a brief overview of quantum cryptography and terms in section 4.1 along with Brassard and Bennett’s protocol. It will then cover the differences between quantum cryptography and other types of cryptography in 4.2. We will then present my Quantum Cryptographical protocol in section 4.3. In Section 4.4, we will show the reduction of our protocol to a transmission protocol based on Brassard and Bennett’s key exchange protocol and then of that protocol to a 15% corruption level (on average) text file. In 4.5, we offer some suggestions to make Quantum Cryptography safer.

4.1 Overview of Quantum Cryptography and Terms

Quantum cryptography, an encryption process the author first learned about in the “Introduction to Analysis of Algorithms” course, relies on the exploitation of quantum

mechanical phenomena such as the uncertainty principle or the violation of a Bell inequality to encode data in a way that is statistically very secure from eavesdroppers, or unwanted listeners[1]. Quantum encryption uses light photons to send bits of information. Light photons, like other traveling particles or waves, can be observed to determine their properties. The wavelength, amplitude, and intensity of a photon of light can all be measured without changing the photon. However, it is theoretically impossible to measure the polarization of a light photon without the possibility of changing the polarization of the photon, unless the measurer knows exactly what polarization the photon already has. The measurement is only precise if the photon and filtering detector are perfectly aligned.

At this point in time, quantum cryptography can only be practiced over relatively short distances, and has difficulty in trying to send a message through an area of other light photons. For the most part it is still in its infancy [7].

In quantum cryptography, a sender (traditionally called Alice) and receiver (traditionally called Bob) agree upon a random pattern of photon polarization orientations allowed within an agreed-upon protocol to send and receive light photons. This random pattern becomes an 'encryption key' between a sender and a receiver. Once in place, the encryption key can be used to encrypt and pass information via a transmission protocol from the sender to the receiver with an incredibly small chance of being successfully eavesdropped upon by a third party (traditionally called Eve) [2].

The first quantum cryptography protocol was developed by Gilles Brassard and Charles Bennett to transmit a secure key. Their protocol stipulated that Alice would send bits to Bob in either a horizontal/vertical orientation or a diagonal-left/diagonal-right

orientation. Alice sends bits, choosing orientations at random, Bob observes orienting receiver at random. Alice and Bob then communicate over a line of communication (separate from the quantum encryption line) about the orientations of the bits during the random bit sending, either horizontal/vertical or diagonal left/diagonal right. Because they were only working with two orientations, about half the time the orientations of Alice and Bob would be mismatched and discarded, but of the times that they both shared the same orientation they would use the bits that were sent and received during those exchanges. They would not, however, discuss what the bit was during those in-phase communications. They would then use the in-phase bits as an encryption key and continue communicating using this encryption key.

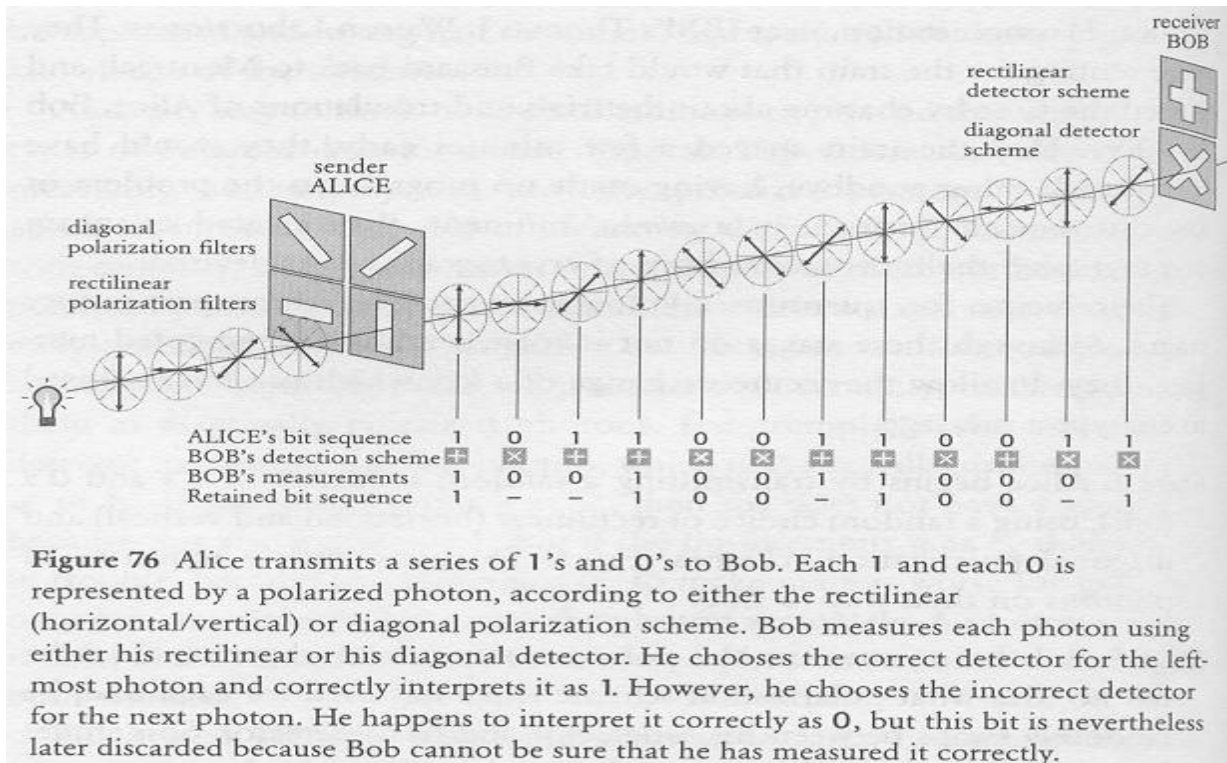


Figure 4.1 Brassard and Bennett's Key Exchange: Figure taken from [7]

4.2 Quantum Cryptography vs. Other types of Cryptography

The major difference between quantum cryptography and other types of cryptography is in their respective regard for eavesdroppers (Eve). In the more commonly used encryption protocols such as RSA public key cryptography, (the protocol used by most secure internet sites) the idea is that any message that is sent or received could, in fact, be intercepted by Eve without Alice or Bob ever knowing [2]. Because of this, Alice and Bob have to rely on the strength of the encryption they use to deter Eve from trying to gain the information. Eve would have to spend countless years trying to use 'brute force' to open even the most basic RSA public key cryptography used by any Internet website.

However, quantum cryptography uses a different strength to prevent Eve from obtaining the message. Quantum cryptography uses photons of light to send its message. It also is theoretically impossible to measure the polarization of these photons without possibly changing them, unless the original orientation is known. Because Alice and Bob can agree upon an encryption key without letting Eve know what the encryption key is, Eve has no way of knowing how to orient the photon detectors as the bits are being sent. Should Eve eavesdrop over future transmissions she will essentially be found out, because she is altering the photons. That said, one could easily imagine a situation where Alice and Bob set up an encryption key using the quantum channel and then leave it open to communicate on it later in the day or week. In this scenario Alice and Bob would not be expecting updates at any particular moment and so if Eve could intercept and decipher

a message being sent from Alice to Bob she could effectively re-encode it and send it back on to Bob without either of them being the wiser this is a Man-in-Middle attack [4].

4.3 Mock Protocol

My claim is that if Alice and Bob did in fact use their new encryption key to devised a pattern of orientations of the polarization filters on the quantum channel (a transmission protocol) then their ensuing communications could be reduced to a corrupted text problem where (on average) 15% of the bits are corrupted.

To demonstrate this, we initially thought about using Brassard and Bennett's two-filter orientation, as a transmission protocol in addition to a key generating protocol, which we would attempt to defeat. However, by using Brassard and Bennett's original protocol as a template, we set out to devise a longer protocol that could be established by using two bits from the key to establish every one-transmission orientation. In this protocol, Alice and Bob would switch back and forth between not two, but four possible filter orientations when sending bits. Each pair of key bits could be used as an indicator of both vertical/horizontal vs. diagonal detector orientation and also zero/one receiving orientation. This protocol would effectively give Eve a 50/50 probability of receiving any one single bit of information correctly.

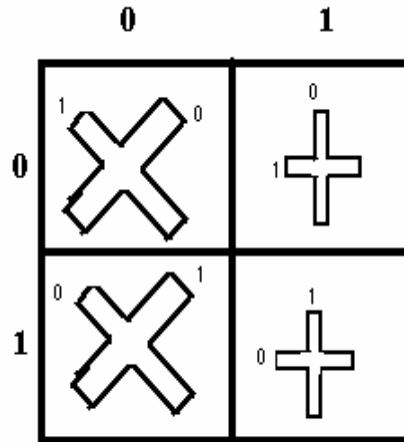


Figure 4.2 Translation scheme: 2-bits encryption key per 1-bit transmission

4.4 Susceptibility to Corrupted File Recovery

Still, this revised protocol, which offers a 50% overall probability of correctly receiving and identifying any given bit in the file, in comparison with the eighty-five/fifteen overall probability, as will be explained below, of the more simple Brassard and Bennett based transmission protocol. This new transmission protocol is susceptible to analysis that on average will yield an eighty-five percent correct bit reception.

Polarized light obeys the laws of physics. Specifically, if there is a photon and a filter that are not in perfect alignment there is a relationship between the size of the angle between the filter and the photon's polarization and the probability that the photo with go through the filter. The precise formula is:

$$I = \cos^2 \Theta ,$$

where I is the probability of the photon's successful passage through the filter [8], and, Θ is the angle between the photon and the filter.

This means that if Eve had a horizontal/vertical filter in place when a diagonal photon came through then Eve would have a $\cos(45 \text{ degrees})^2$ (or 50%) chance of correctly identifying the bit. However, if Eve tilted her filter to an angle in-between the horizontal/vertical and the diagonal filter, then she would have a $\cos(22.5)^2$ (or 85.36%) chance of getting every bit sent.

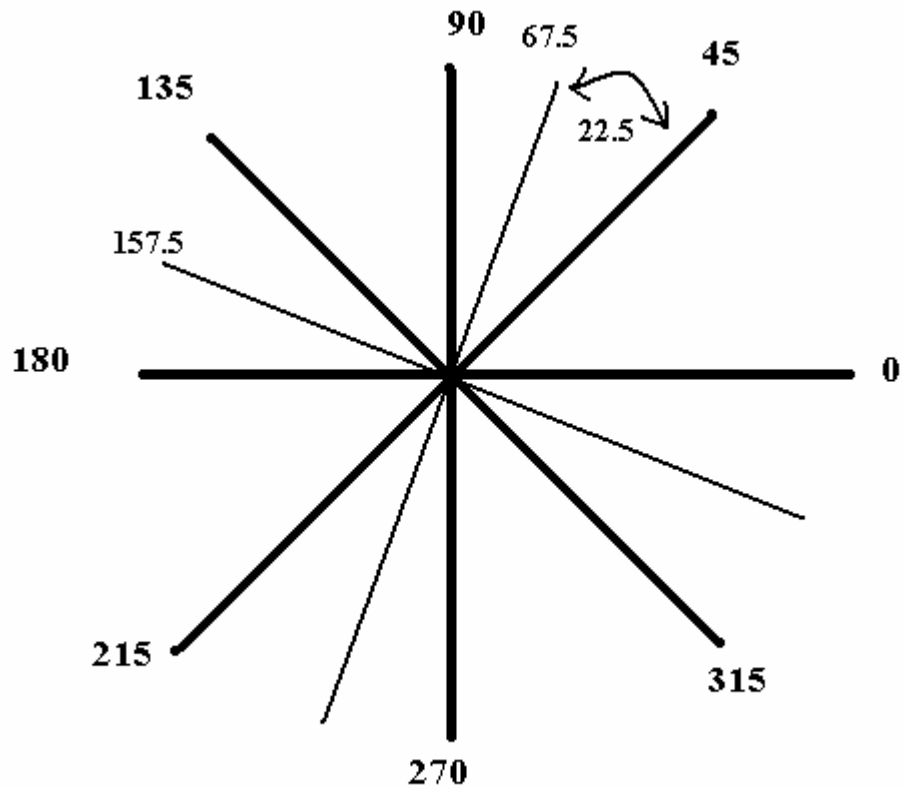


Figure 4.3 Vertical/Horizontal – Diagonal photon splitting orientation

By doing this, Eve could receive 85% of the bits correctly for a transfer scheme based on Brassard and Bennett's two orientations. (i.e., a situation where 85% of the bits are correct and 15% of the bits are incorrect, or rather 15% corruption). Thus, since the problem can be reduced to a 15% corrupted text file reconstruction, the encryption can be cracked.

To defeat the four orientations scheme explained earlier, Eve needs to know one more piece of information. Namely, she has to know the length of the encryption key that Alice and Bob are using. This is realistic because Alice and Bob had to speak over a communication line that was not the quantum line, in order to discuss the key's orientations. So all Eve has to do is monitor that communication and see how many orientations Alice and Bob have in common.

Knowing this piece of information, Eve can monitor the first of every eight bits transmitted and know that each first bit should be zero 85% of the time and one 15% of the time. This is because (as discussed in chapter 2) the high level bit is always zero in our predefined set of input texts. Unless the key is perfectly divisible by 8 or vice-versa, the top order bit will wander over different positions in the key, providing a means to detect the encoding relative to the orientation of the receiver (one up, or zero up) is. Should the key be too short or for some other reason the bit columns are not exposed top order bit method, the rest of the bits can be guessed. There should only be a few combinations of the bit columns (as each time the key repeats all values deciphered by that position in the key will be of the same orientation).

Essentially if Eve is given this harder version of the Brassard and Bennett based transmission protocol, she can still get 85% of the bits correct. This is because all values interpreted by the same position of the key will always have the same orientation. By lining up all values interpreted by the same position of the key, Eve will have all values for that position either be approx 85% correct or 15% correct. By using the top bit analysis discussed above, she can flip all bits received from one key position that is shown to have 15% of its bits correct (by flipping these bits 15% correct will switch to 85% correct).

4.5 Suggested Protocol Modifications

The first and most obvious thing to change about the protocol is to not use the quantum channel after it has been used to make a key. Eve can't know what the key is, but because Eve could know the length of the key, it would be possible for her to reduce a transmission from Alice to Bob into a 15% corrupted text problem.

Another obvious change to the protocol would be to not use ASCII to transmit any information over the channel. It is only possible to decrypt the transmission because they are encoded in ASCII. A more compact coding scheme would aid in reducing this decryption possibility.

Finally, Alice and Bob could use very long keys in conjunction with the four orientations method to prevent Eve from getting enough statistical information to decode the transmitted file.

Chapter 5

Conclusion

Corruption is a problem because whenever computer information is transmitted from one place to another it is subject to random noise, and the bits conveying the information may be received as the opposite value from their intended one. This leads to various problems in the computer world such as corrupt files, noise in audio channels, etc. This thesis proposed a new approach to solving these problems by being able to reconstruct a file after you have obtained it partially corrupted. It is our belief that all of this research is new and, barring an unfound research document surfacing, it is also in a totally new line of research.

This thesis reconstructs corrupted files through the use of genetic algorithms and analysis of ASCII text. A genetic algorithm is a heuristic process that uses the idea of genetic recombination and mutation to produce ‘populations’ of solutions to a problem that ‘mate’ with each other to produce new (and over a sequence of generations) better solutions. Also part of this thesis analysis is the use of the ASCII encoding, which this thesis has shown to have intrinsic qualities that aid in reconstructing texts.

Specifically, through analysis of ASCII we successfully broke files corrupted at a 15% rate into their sentences. Thus, breaking corrupted files into sentences can occur very cleanly. Additionally, in doing our experimentation we found that the profile of an end of sentence is sufficiently noticeable to have the algorithm identify them without full

optimization of algorithm parameters. Moreover, the genetic algorithm used by this thesis is able to reduce a 15% overall corruptness level of an input specimen back towards its original words. Most strikingly, the genetic algorithm returns a specimen with all words. Also, a significant portion of the remaining corruptedness of a specimen is localized to false positive spaces. All of this allows for a percentage corruption drop from an initial 15% corruption down to 7.5% corruption, and, should the false positive spaces be removed, a corruption rate below 5%

One application of this genetic algorithm is deciphering quantum encryption transmissions. This thesis has shown that simple transmission protocols based on quantum cryptography are not as secure as simple analysis might imply.

This thesis showed, using two different protocols, how the problem of decrypting a quantum encryption transmission, reduces to the problem of reconstructing corrupted text. The first protocol, based on Brassard and Bennett's quantum key encryption method, can be reduced by orienting a detector between those of the two positions being used by the sender and receiver. The second protocol, slightly more complex, can be reduced to the first protocol by knowing the key length, and making use of the fact that ASCII's top order bit is constant over our assumed set of input.

One area for future research, to be incorporated into a genetic algorithm, would be the use of context specific information. Also, future genetic algorithms could incorporate grammar checking into its fitness function. Still another future genetic algorithm could look at using common word pairings and syllables. And, finally, one future genetic algorithm might look at incorrectly spelled words.

Taken in total, this thesis makes strong initial strides into this area of research.

Bibliography

- [1] Barnett, S. M Huttner, B and Phoenix, S. J. D., “Eavesdropping strategies and rejected data protocols in quantum cryptography”, *Journal of Modern Optics*, vol. 40, no.12, December 1993, pp. 2501 – 2513.
- [2] Cormen, T.H., C.E. Leiserson, R.L. Rivest, C. Stein. 2001, c1990. *Introduction to Algorithms Second Edition*. Cambridge, Massachusetts. The MIT Press.
- [3] Crystal, David, 1987. *The Cambridge Encyclopedia of Language*. New York, NY. Cambridge University Press.
- [4] Ekert, A. K., Huttner, B., Palma, G. M. and Peres, A., “Eavesdropping on quantum cryptosystems”, *Physical Review A*, submitted, vol. 50, no. 2, August 1004, pp. 1047-1056.
- [5] Horowitz, Paul. 1994, c1989. *The Art of Electronics*. New York, NY. Cambridge University Press.
- [6] Man, K.F., K.S. Tang, and S. Kwong. 1999. *Genetic Algorithms*. London, England. Springer-Verlag London Limited.
- [7] Singh, Simon. 1999. *The Code Book*. New York, NY. Anchor Books.
- [8] Tipler, Paul A. 1990. *Physics for Scientists and Engineers*. New York, NY. W.H. Freeman and Company/Worth Publishers.

Appendix A

Corruption Implementation

```
*****Corrupter.h*****

// Corrupter.h

// JPhillips Honors Thesis, Fall 2000

// specification file for Corrupter class

#ifndef CORRUPTER_H
#define CORRUPTER_H
#include <string>
using namespace std;

class Corrupter {
public:
    Corrupter(void);
        // Purpose: create corrupter

    ~Corrupter(void);
        // Purpose: free dynamically allocated memory. Delete corrupter.

    void Setinfile(void);
        // Purpose: ask user to specify the file to corrupt. Set infile to this
        // value
        // Pre: none
        // Post: infile is set to specified value of input from user.

    void Setoutfile(void);
        // Purpose: ask user to specify the file to print corrupted file to. Set
        // outfile to this value
        // Pre: none
        // Post: outfile is set to specified value of input from user.

    void Setpercort(void);
        // Purpose: ask user to set the percent of corruption for the file
        // Pre: none
        // Post: percort is set to specified value of input from user.

    char* Getinfilename(void);
        // Purpose: allow user to check what file is corrupted.
```

```

// Pre: infile is initialized.
// Post: return infile.

char* Getoutfilename(void);
// Purpose: allow user to check what file is being printed to.
// Pre: outfile is initialized.
// Post: return outfile.

int Getpercortext(void);
// Purpose: allow user to check what percortext value is set to.
// Pre: percortext is initialized.
// Post: return percortext.

char* Getcortext(void);
// Purpose: allow cortext array to be transferred to other location.
// Pre: cortext is initialized.
// Post: return cortext.

void Corruptinfile(void);
// Purpose: populate the characters of infile with random percortext% corrupted
// and store in character array cortext.
// Pre: percortext and infile are initiated.
// Post: cortext contains the characters of infile with a random percortext% of
// them corrupted

void Cortooutfile(void);
// Purpose: print corrupted text (cortext) array to output file (outfile).
// Pre: cortext is populated with a corrupted version of infile.
// Post: file specified as outfile contains cortext.
void Ortooutfile(void);
// Purpose: print original text (original) array to output file (outfile).
// Pre: original is populated with the original version of infile.
// Post: file specified as outfile contains original.
void Cortoscreen(void);
// Purpose: print corrupted text (cortext) array to screen.
// Pre: cortext is populated with a corrupted version of infile.
// Post: cortext has been output to screen, however characters that the
// screen can't handle are printed as 'X'.
void Ortoscreen(void);
// Purpose: print original text (original) array to screen.
// Pre: original is populated with a original version of infile.
// Post: original has been output to screen.

```

private:

```

char Makemask();
// Purpose: Create a random char(8 bits long) with percortext% 1's and the rest
// 0's.
// Pre: percortext must be initiated.
// Post: returns a char that has about percortext% of its bits randomly set to

```

```

// 1 (one).

int percor;
char original[10000];
char cortext[10000];
int ctlength;
char infile[25];
char outfile[25];

};

#endif

*****Corrupter.cc*****

```

```

#include <string>
#include <fstream>
#include <ostream>
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <cmath>
#include "Corrupter.h"

```

```

using namespace std;

```

```

Corrupter::Corrupter(void)
// Purpose: create corrupter
{
    percor = 0;
    ctlength = 0;
}

```

```

Corrupter::~Corrupter(void)
// Purpose: free dynamically allocated memory. Delete corrupter.
{
    delete infile;
    delete outfile;
    delete original;
    delete cortext;
}

```

```

void Corrupter::Setinfile(void)
// Purpose: ask user to specify the file to corrupt. Set infile to this
// value
// Pre: none

```

```

    // Post: infile is set to specified value of input from user.
    {
    cout << "Please enter the name of the file you wish to corrupt. >> ";
    cin >> infile;
    }

void Corrupter::Setoutfile(void)
    // Purpose: ask user to specify the file to print corrupted file to. Set
    // outfile to this value
    // Pre: none
    // Post: outfile is set to specified value of input from user.
    {
    cout << "Please enter the name of the file you wish to print to. >>";
    cin >> outfile;
    }

void Corrupter::Setpercort(void)
    // Purpose: ask user to set the percent of corruption for the file
    // Pre: input value must be between 0 and 100.
    // Post: percort is set to specified value of input from user.
    {
    cout << "Please enter the percent of the file you wish to corrupt. Must be
between 0 and 100. >>";
    cin >> percort;
    }

char* Corrupter::Getinfilename(void)
    // Purpose: allow user to check what file is corrupted.
    // Pre: infile is initialized.
    // Post: return infile.
    {
    return infile;
    }

char* Corrupter::Getoutfilename(void)
    // Purpose: allow user to check what file is being printed to.
    // Pre: outfile is initialized.
    // Post: return outfile.
    {
    return outfile;
    }

int Corrupter::Getpercort(void)
    // Purpose: allow user to check what percort value is set to.
    // Pre: percort is initialized.
    // Post: return percort.
    {
    return percort;
    }

char* Corrupter::Getcortext(void)
    // Purpose: allow cortext array to be transferred to other location.
    // Pre: cortext is initialized.
    // Post: return cortext.

```

```

{
    return cortext;
}

```

```

void Corrupter::Corruptinfile(void)
    // Purpose: populate the characters of infile with random percors% corrupted
    // and store in character array cortext.
    // Pre: percors and infile are initiated.
    // Post: cortext contains the characters of infile with a random percors% of
    // them corrupted
{
    ctlength = 0;           //counter for file-array
    char c1;               //read in value
    char c2;               //corrupting mask
    ifstream file(infile);
    while(file.get(c1)){
        c2 = Makemask();
        cortext[ctlength] = (char)(c1 ^ c2);
        original[ctlength] = (char)c1;
        ctlength++;
    }
}

```

```

void Corrupter::Cortoofile(void)
    // Purpose: print corrupted text (cortext) array to output file (outfile).
    // Pre: cortext is populated with a corrupted version of infile.
    // Post: file specified as outfile contains cortext.
{
    int n = 0;
    ofstream togofile(outfile);
    while(n < ctlength){
        togofile << cortext[n];
        n++;
    }
}

```

```

void Corrupter::Ortoofile(void)
    // Purpose: print original text (original) array to output file (outfile).
    // Pre: original is populated with the original version of infile.
    // Post: file specified as outfile contains original
{
    int n = 0;
    ofstream togofile(outfile);
    while(n < ctlength){
        togofile << (char)original[n];
        n++;
    }
}

```

```

void Corrupter::Cortoscreen(void)
    // Purpose: print corrupted text (cortext) array to screen.
    // Pre: cortext is populated with a corrupted version of infile.
    // Post: cortext has been output to screen, however characters that the
    // screen can't handle are printed as 'X'.
{

```

```

int n = 0;
while(n < ctextlength){
    if(cortex[n] >= '\40' && cortex[n] <= '\177'){
        cout << cortex[n];}
    else if (cortex[n] > '\1770'){
        cout << 'Y';}
    else cout << 'X';
    n++;
}
cout << endl;
}
void Corrupter::Ortoscreen(void)
// Purpose: print original text (original) array to screen.
// Pre: original is populated with a original version of infile.
// Post: original has been output to screen.
{
    int n = 0;
    while(n < ctextlength){
        cout << (char)original[n];
        n++;
    }
    cout << endl;
}

char Corrupter::Makemask()
// Purpose: Create a random char(8 bits long) with percor% 1's and the rest
// 0's.
// Pre: percor must be initiated.
// Post: returns a char that has about percor% of its bits randomly set to
// 1 (one).
{
    char randomval = '\0';
    int i;
    for(int count = 0; count<8; count++){
        i = rand() % 100;
        if(i < percor){
            (unsigned)randomval = (unsigned)randomval * 2;
            (unsigned)randomval = (unsigned)randomval + 1;
        }
        else
            (unsigned)randomval = (unsigned)randomval << 1;
    }
    return randomval;
}

```

Appendix B

Text Analysis Implementation

```
*****Manipulator.h*****

// Manipulator.h

// JPhillips Honors Thesis, Fall 2003

// specification file for Corrupter class

#ifndef MANIPULATOR_H
#define MANIPULATOR_H
#include <string>
using namespace std;

class Manipulator {
public:
    Manipulator(void);
        // Purpose: create Manipulator

    ~Manipulator(void);
        // Purpose: free dynamically allocated memory. Delete Manipulator.

    void Setinfile(void);
        // Purpose: ask user to specify the file to manipulate. Set infile to
        // thisvalue
        // Pre: none
        // Post: infile is set to specified value of input from user.

    void Setoutfile(void);
        // Purpose: ask user to specify the file to print manipulated file to.
        // Setoutfile to this value
        // Pre: none
        // Post: outfile is set to specified value of input from user.

    char* Getinfilename(void);
        // Purpose: allow user to check what file is manipulated.
        // Pre: infile is initialized.
        // Post: return infile.

    char* Getoutfilename(void);
        // Purpose: allow user to check what file is being printed to.
        // Pre: outfile is initialized.
        // Post: return outfile.
```

```

float Getsamplepercor(void);
// Purpose: allow user to check what samplepercor value is set to.
// Pre: samplepercor is initialized.
// Post: return samplepercor.

char* Getmantext(void);
// Purpose: allow mantext array to be transfered to other location.
// Pre: mantext is initialized.
// Post: return mantext.

void Toporderbit(void);
// Purpose: To make all top order bits 0 and count all the times it is a 1
// Pre: Must not have been run yet. infile is initialized.
// Post: saplepercor is the percent of top order bits corrupted /sample
// size. Also, Mantext is populated with all 0's in top order bitplaces.

void Mantoofile(void);
// Purpose: print manipulated text (mantext) array to output file
// (outfile).
// Pre: cortext is populated with a corrupted version of infile.
// Post: file specified as outfile contains cortext.
void Ortoofile(void);
// Purpose: print original text (original) array to output file (outfile).
// Pre: original is populated with the original version of infile.
// Post: file specified as outfile contains original.
void Mantoscreen(void);
// Purpose: print manipulated text (mantext) array to screen.
// Pre: cortext is populated with a corrupted version of infile.
// Post: cortext has been output to screen, however characters that the
// screen can't handle are printed as 'X'.
void Ortoscreen(void);
// Purpose: print original text (original) array to screen.
// Pre: original is populated with a original version of infile.
// Post: original has been output to screen.
void Percentlevs(char inchar);

//int Isclosesto(char inchar1, char inchar2);
// Purpose: Compare two chars and determine how close (bitwise) they are
// Pre: inchar1 & inchar2 are char values
// Post: return the number of bits difference between inchar1 & inchar2
void Findsentends();
private:

// bool Issentend(char inchar1, char inchar2, char inchar3, char inchar4);
// Purpose: Identify if inchar1-4 are the end of a sentance.
// Pre: samplepercor is populated & inchar1-4 are char values.
// Post: returns true if inchar1-4 are sufficiently close to a sentance

```

```

// end. Also prints the end type.

bool lsspace(char inchar);
// Purpose: Identify if inchar is/ or was close to a space.
// Pre: mantext is populated with a manipulated version of infile.
// Post: lsspace returns true if inchar was close to a space.

int enterchar[8]; // this holds the bit version of the inchar
int comparcharlevels[9]; //this holds the number of good bits at a level

double samplepercor;
char original[10000];
char mantext[10000];
int mlength;
char infile[25];
char outfile[25];

};

#endif

```

*****Manipulator.cc*****

```

#include <string>
#include <fstream>
#include <ostream>
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <cmath>
#include "Manipulator.h"

using namespace std;

Manipulator::Manipulator(void)
// Purpose: create Manipulator
{
    samplepercor = 0;
    mlength = 0;
}

Manipulator::~Manipulator(void)
// Purpose: free dynamically allocated memory. Delete Manipulator.
{
    delete infile;
    delete outfile;
    delete original;
    delete mantext;
}

```

```
}
```

```
void Manipulator::Setinfile(void)
    // Purpose: ask user to specify the file to manipulate. Set infile to
    // thisvalue
    // Pre: none
    // Post: infile is set to specified value of input from user.
{
    cout << "Please enter the name of the file you wish to manipulate. >> ";
    cin >> infile;
}
```

```
void Manipulator::Setoutfile(void)
    // Purpose: ask user to specify the file to print manipulated file to.
    // Setoutfile to this value
    // Pre: none
    // Post: outfile is set to specified value of input from user.
{
    cout << "Please enter the name of the file you wish to print to. >>";
    cin >> outfile;
}
```

```
char* Manipulator::Getinfilename(void)
    // Purpose: allow user to check what file is manipulated.
    // Pre: infile is initialized.
    // Post: return infile.
{
    return infile;
}
```

```
char* Manipulator::Getoutfilename(void)
    // Purpose: allow user to check what file is being printed to.
    // Pre: outfile is initialized.
    // Post: return outfile.
{
    return outfile;
}
```

```
float Manipulator::Getsamplepercor(void)
    // Purpose: allow user to check what samplepercor value is set to.
    // Pre: samplepercor is initialized.
    // Post: return samplepercor.
{
    return samplepercor;
}
```

```
char* Manipulator::Getmantext(void)
    // Purpose: allow mantext array to be transfered to other location.
    // Pre: mantext is initialized.
    // Post: return mantext.
{
    return mantext;
}
```

```

void Manipulator::Toporderbit(void)
// Purpose: To make all top order bits 0 and count all the times it is a 1
// Pre: Must not have been run yet. infile is initialized.
// Post: sampleperc is the percent of top order bits corrupted /sample
// size. Also, Mantext is populated with all 0's in top order bitplaces.
{
    mtlength = 0;           //counter for file-array
    char c1, c2;
    int n;
    int numcor = 0;
    ifstream file(infile);
    sampleperc = 0;

    cout << "working loop" << endl;
    while(file.get(c1)){
        c2 = c1;

/*
        if((c1 >= '\40' && c1 < '\177')||(c1=='\11' ||c1=='\12')){
            cout << c1;}
        else cout << "X";
*/

/*
        if((unsigned char)c1 >= '\200'){
            (unsigned char)c2 = (unsigned char)c1%'\200';
            numcor++;
            cout <<"Numcor is " <<numcor << endl;
        }
THIS IS THE ORIGINAL CODE */
        if((unsigned char)c1 >= 0200){
            (unsigned char)c2 = (unsigned char)c1%0200;
            numcor++;
            //cout <<"Numcor is " <<numcor << endl;
        }

        if((c2 >= '\40' && c2 < '\177')||(c2=='\11' ||c2=='\12')){
            cout << c2;
        }
        else cout << "X";

        mantext[mtlength] = (char)c2;
        original[mtlength] = (char)c1;
        mtlength++;
        //cout <<"mtlength is " <<mtlength << " for "<< (unsigned char)c1<<
endl;
    }

    if(numcor!=0){
        sampleperc = (double)numcor/(double)mtlength;
    }
}

```

```

// cout << endl<< " SAMPLEPERCOR = " << samplepercor << "numcor =
// "<<(double)numcor <<"mtlength = " <<(double)mtlength <<" SAMPLEPERCOR =
// " << (double)numcor/(double)mtlength << endl;

}

void Manipulator::Mantoofile(void)
// Purpose: print manipulated text (mantext) array to output file
// (outfile).
// Pre: cortext is populated with a corrupted version of infile.
// Post: file specified as outfile contains cortext.
{
int n = 0;
ofstream tofile(outfile);
while(n < mtlenght){
tofile << mantext[n];
n++;
}
}

void Manipulator::Ortoofile(void)
// Purpose: print original text (original) array to output file (outfile).
// Pre: original is populated with the original version of infile.
// Post: file specified as outfile contains original
{
int n = 0;
ofstream tofile(outfile);
while(n < mtlenght){
tofile << (char)original[n];
n++;
}
}

void Manipulator::Mantoscreen(void)
// Purpose: print manipulated text (mantext) array to screen.
// Pre: cortext is populated with a corrupted version of infile.
// Post: cortext has been output to screen, however characters that the
// screen can't handle are printed as 'X'.
{
int n = 0;
while(n < mtlenght){
if((mantext[n] >= '\40' && mantext[n] <=
'\177')||/**/(mantext[n]=='\11'||mantext[n]=='\12')){
cout << mantext[n];
}
else cout << 'X';
n++;
}
cout << endl;
}

void Manipulator::Ortoscreen(void)
// Purpose: print original text (original) array to screen.
// Pre: original is populated with a original version of infile.

```

```

// Post: original has been output to screen.
{

ofstream tofile("STATSORGOUTPUT.txt");
int n = 0;
while(n < mlength){
    cout << (char)original[n];
    n++;
}
cout << endl;

}

```

```

bool Manipulator::Isspace(char inchar)
// Purpose: Identify if inchar is/ or was close to a space.
// Pre: mantext is populated with a manipulated version of infile.
// Post: Isspace returns true if inchar was close to a space.
{
    bool value = false;

    return value;
}

```

```

double Topower(double a,int b){
    double c = 0;

    if(b > 0){
        c=a;
        b--;
        while (b>0){
            c = c * a;
            b--;
        }
    }
    else if(b == 0)
    {
        c = 1;
    }
    return c;
}

```

```

int Factorial(int a){
    int b = 0;
    if (a > 0){
        b=a;
        a--;
        while(a > 0){
            b = b * a;
            a--;
        }
    }
    else if (a == 0){
        b = 1;
    }
}

```

```

    return b;
}

void Manipulator::Percentlevs(char inchar)
{
    double x=0;
    double y=0;

    int n = 0;
    if((inchar/128)==1){enterchar[0]=1;}else enterchar[0]=0;inchar=inchar%128;
    if((inchar/64)==1){enterchar[1]=1;}else enterchar[1]=0;inchar=inchar%64;
    if((inchar/32)==1){enterchar[2]=1;}else enterchar[2]=0;inchar=inchar%32;
    if((inchar/16)==1){enterchar[3]=1;}else enterchar[3]=0;inchar=inchar%16;
    if((inchar/8)==1){enterchar[4]=1;}else enterchar[4]=0;inchar=inchar%8;
    if((inchar/4)==1){enterchar[5]=1;}else enterchar[5]=0;inchar=inchar%4;
    if((inchar/2)==1){enterchar[6]=1;}else enterchar[6]=0;inchar=inchar%2;
    if((inchar/1)==1){enterchar[7]=1;}else enterchar[7]=0;

    for (int s =0; s < 9; s++){

        comparcharlevels[s] = 0;
    }

    for(int a = 0; a < 2; a++){
        for(int b = 0; b < 2; b++){
            for(int c = 0; c < 2; c++){
                for(int d = 0; d < 2; d++){
                    for(int e = 0; e < 2; e++){
                        for(int f = 0; f < 2; f++){
                            for(int g = 0; g < 2; g++){
                                for(int h = 0; h < 2; h++){
                                    if(a==0 && ((b==0 && c==1)||b==1)
                                        && !(a==0 && b==1 && c==1 && d==1 &&
e==1 && f==1 && g==1 && h==1)) ){

                                        n=0;
                                        if(enterchar[0]!=a){n++;}
                                        if(enterchar[1]!=b){n++;}
                                        if(enterchar[2]!=c){n++;}
                                        if(enterchar[3]!=d){n++;}
                                        if(enterchar[4]!=e){n++;}
                                        if(enterchar[5]!=f){n++;}
                                        if(enterchar[6]!=g){n++;}
                                        if(enterchar[7]!=h){n++;}
                                        comparcharlevels[n]++;

                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }

    for(int i = 0; i < 9; i++){
        x =
(Factorial(8)/(Factorial(8-i)*Factorial(i)))*Topower(samplepercor,i);
        y = Topower(1-samplepercor,8-i);
        cout << "Level"<< i <<"-"<< i <<"bits changed "<< x*y << endl;
        cout << "There are " << (Factorial(8)/(Factorial(8-i)*Factorial(i)))
            << " theoretical mappings of which" <<comparcharlevels[i]
            <<" are actually possibilities " << endl;
        cout << endl << endl;
    }
}

int Iscloseto(char inchar1, char inchar2)
// Purpose: Compare two chars and determine how close (bitwise) they are
// Pre: inchar1 & inchar2 are char values
// Post: return the number of bits difference between inchar1 & inchar2
{
    int count = 0;

    if((inchar1/128)!=inchar2/128){count++;}
    inchar1=inchar1%128;inchar2=inchar2%128;
    if((inchar1/64)!=inchar2/64){count++;}
    inchar1=inchar1%64;inchar2=inchar2%64;
    if((inchar1/32)!=inchar2/32){count++;}
    inchar1=inchar1%32;inchar2=inchar2%32;
    if((inchar1/16)!=inchar2/16){count++;}
    inchar1=inchar1%16;inchar2=inchar2%16;
    if((inchar1/8)!=inchar2/8){count++;}
    inchar1=inchar1%8;inchar2=inchar2%8;
    if((inchar1/4)!=inchar2/4){count++;}
    inchar1=inchar1%4;inchar2=inchar2%4;
    if((inchar1/2)!=inchar2/2){count++;}
    inchar1=inchar1%2;inchar2=inchar2%2;
    if((inchar1/1)!=inchar2/1){count++;}

    return count;
}

bool Issentend(char inchar1, char inchar2, char inchar3, char inchar4, double
samplepercor,ofstream & togofile)
// Purpose: Identify if inchar1-4 are the end of a sentence.
// Pre: samplepercor is populated & inchar1-4 are char values.
// Post: returns true if inchar1-4 are sufficiently close to a sentence
// end. Also prints the end type.
{
    bool outbool = false;

    const int c1 = 16;
    const int c2 = 16; //const weight values
    const int c3 = 16;
    const int c4 = 7;
    const int c5 = 3;

```

```

const float v1 = 0.775; // . //const measuring values
const float v2 = 0.95; // !
const float v3 = 0.95; // ?

int v1count = 0;
int v2count = 0;
int v3count = 0;

const int c1bits = 7;
const int c2bits = 7;
const int c3bits = 7;
const int c4bits = 2;
const int c5bits = 1;

const int totalbits = c1bits + c2bits + c3bits + c4bits + c5bits;

float b1 = 0;
float b2 = 0;
float b3 = 0;
float b4 = 0;
float b5 = 0;

float q1 = 0.0;
float q2 = 0.0;
float q3 = 0.0;

b2 = Iscloseto('\40',inchar2);
b3 = Iscloseto('\40',inchar3);

//*****
// b4 and b5
//*****

if((inchar4/128)!=('\101'/128)){b4++;}
if(((inchar4%128)/64)!=('\101'%64)/64){b4++;}

if(((inchar4%32)!=\100'%32)&&
(inchar4%32!=\134'%32)&&
(inchar4%32!=\135'%32)&&
(inchar4%32!=\136'%32)&&
(inchar4%32!=\137'%32)){b5++;}

//*****
// end of b4 and b5
//*****

//*****
// b1's and q1,q2,q3
//*****

b1 = Iscloseto('\56',inchar1);
q1 = ((b1*c1)+(b2*c2)+(b3*c3)+(b4*c4)+(b5*c5)/
((c1bits*c1)+(c2bits*c2)+(c3bits*c3)+(c4bits*c4)+(c5bits*c5));
togofile << b1;

```

```

b1 = Iscloseto('\41',inchar1);
q2 = ((b1*c1)+(b2*c2)+(b3*c3)+(b4*c4)+(b5*c5))/
      ((c1bits*c1)+(c2bits*c2)+(c3bits*c3)+(c4bits*c4)+(c5bits*c5));
togofile << b1;

b1 = Iscloseto('\77',inchar1);
q3 = ((b1*c1)+(b2*c2)+(b3*c3)+(b4*c4)+(b5*c5))/
      ((c1bits*c1)+(c2bits*c2)+(c3bits*c3)+(c4bits*c4)+(c5bits*c5));
togofile << b1;

togofile << b2;
togofile << b3;
togofile << b4;
togofile << b5;
togofile << samplepercor;
togofile << endl;

if(q1 <= 1-v1){
    cout << " Found .__C " << endl;
    outbool = true;
}
if(q2 <= 1-v2){
    cout << " Found !__C " << endl;
    outbool = true;
}
if(q3 <= 1-v3){
    cout << " Found ?__C " << endl;
    outbool = true;
}

return outbool;
}

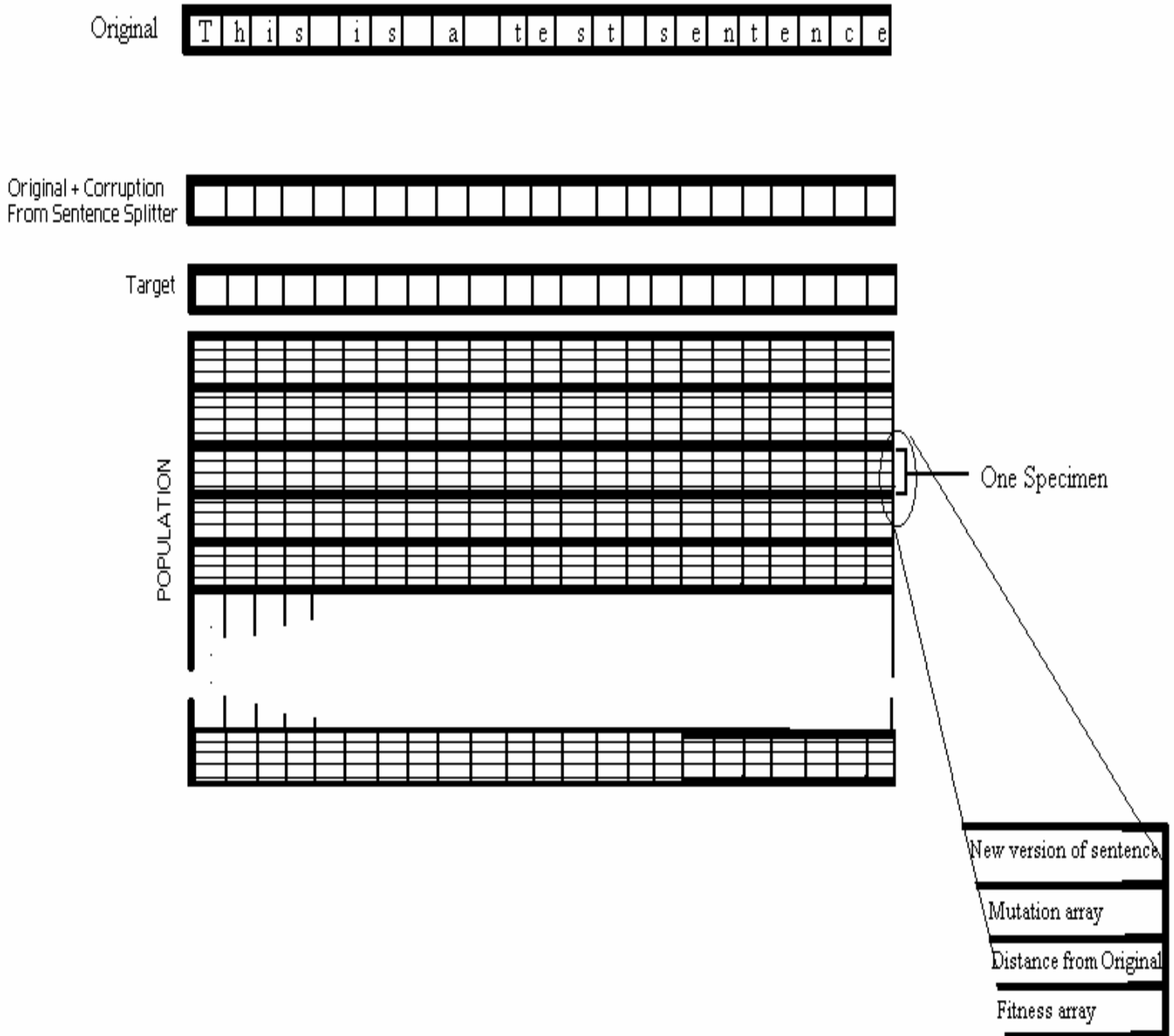
void Manipulator::Findsentends(){
    ofstream togofile("STATSOUTPUT.txt");
    for(int i = 0;i<mtlength-3; i++){
        if(mantext[i] >= '\40' && mantext[i] < '\177'){
            cout << mantext[i]; //<< " finding ...";
        }
        else cout << "X";
    }

    lssentend(mantext[i],mantext[i+1],mantext[i+2],mantext[i+3],samplepercor,togofil
e);
    //cout << " samplepercor is " << samplepercor << endl;
}
}

```

Appendix C

Genetic Algorithm Implementation



*****GenAlg.h*****

```
#ifndef GENALG_H
#define GENALG_H
#include <string>
using namespace std;

struct Child {
    double fitnessvalue;
    double corruptness; //percent of newcorrupt's corruptness from ortext
    char *newcorrupt; //length
    int *distance; //length
    double *fitness; //length
    double *mutation; //lengthd*8
};

class GenAlg {
public:
    //  CONTSTRUCTOR
    GenAlg(int numspec, double percentcor, int gens, string file, double
SMWM,double BGWM,double CORM,double SPM);
    //  DESTRUCTOR
    ~GenAlg(void);

    //  FUNCTIONS DEALING WITH ENTERY TEXT
    void TargetEN(void); //looks at entered text and targets specific bits
    void Randomize(int size); //Creates Randomized Population of Strucks.
    void Test(int size);
    void TestPrint1(void);
    void TestPrint2(void);

    //  FUNCTIONS DEALING WITH ARRAY OF CHILD'S
    void Order(int numofpops, Child *pop, int popstart, int popend);
    //Order array of child's with the best fitness's at
    void Breed(void);
        //Make new Child's from current Child's repopulate
        //array of Childs

    //  FUNCTIONS DEALING WITH STRUCTS
    void Mutate(int inChild, Child *pop);
        //used in the Breed function, this mutates the end
        //results of simple crossover breeding.

    void Distance(int inChild, Child *pop); //compute the distance between the
STRUCT's
        //text and the entered text. (bitwise?) deals with
        //the distance array and the Corruptness member.

    void TargetCH(int inChild, Child *pop); //looks at Struct's text and
targets specific bits to
        //increase likelihood of flipping them. Deals with
        //the Mutation array.

    void Fitness(int inChild, bool Cpop, bool Opop); //computes the fitness
```

of the Struct's text. Deals
//with the Fitness array and the fitnessvalue member

```
bool Dictionary(int count, string word);  
private:
```

```
    //bool Dictionary(int count, string word);  
    int length;//length of characters  
    double percor;  
    char *ortext;//length  
    int *ortextbits;//length*8  
    double *target;//length*8  
    int numloops;  
    int totloops;
```

```
    double SmallwordMult;  
    double BigwordMult;  
    double corruptMult;  
    double spaceMult;
```

```
    int popsize;//size of the population
```

```
    Child *population;  
    Child *childpopulation;  
    Child *newpopulation;
```

```
    char Makemask(int place);
```

```
    int quickprime[23];
```

```
    string in1[41];  
    string in2[449];  
    string in3[2039];  
    string in4[6397];  
    string in5[12973];  
    string in6[23531];  
    string in7[31189];  
    string in8[34871];  
    string in9[34871];  
    string in10[31189];  
    string in11[24151];  
    string in12[18211];  
    string in13[12973];  
    string in14[6397];  
    string in15[5099];  
    string in16[2039];  
    string in17[1493];  
    string in18[797];  
    string in19[337];  
    string in20[79];  
    string in21[41];  
    string in22[11];  
    string in23[7];
```

```
};  
#endif
```

```
*****GenAlg.cc*****
```

```
#include <string>  
#include <fstream>  
#include <ostream>  
#include <iostream>  
#include <cstdlib>  
#include <stdlib.h>  
#include <cmath>  
#include "GenAlg.h"
```

```
using namespace std;
```

```
const string infile1 = "HashDictionary1.txt";  
const string infile2 = "HashDictionary2.txt";  
const string infile3 = "HashDictionary3.txt";  
const string infile4 = "HashDictionary4.txt";  
const string infile5 = "HashDictionary5.txt";  
const string infile6 = "HashDictionary6.txt";  
const string infile7 = "HashDictionary7.txt";  
const string infile8 = "HashDictionary8.txt";  
const string infile9 = "HashDictionary9.txt";  
const string infile10 = "HashDictionary10.txt";  
const string infile11 = "HashDictionary11.txt";  
const string infile12 = "HashDictionary12.txt";  
const string infile13 = "HashDictionary13.txt";  
const string infile14 = "HashDictionary14.txt";  
const string infile15 = "HashDictionary15.txt";  
const string infile16 = "HashDictionary16.txt";  
const string infile17 = "HashDictionary17.txt";  
const string infile18 = "HashDictionary18.txt";  
const string infile19 = "HashDictionary19.txt";  
const string infile20 = "HashDictionary20.txt";  
const string infile21 = "HashDictionary21.txt";  
const string infile22 = "HashDictionary22.txt";  
const string infile23 = "HashDictionary23.txt";
```

```
int Iscloseto(char inchar1, char inchar2);
```

```
GenAlg::GenAlg(int numspec, double percentcor, int gens, string file, double  
SMWM,double BGWM,double CORM,double SPM)  
{  
    string temp;
```

```

ifstream corsent("DECPrepared2.txt");
getline(corsent, temp);
length = temp.length();
ortext = new char[length];
for(int j=0; j<length; j++){
    ortext[j] = (char) temp[j];
}
percor = percentcor;
cout << percentcor<< endl;
// percor = 0.128205;
// cout << 0.128205 << endl;

SmallwordMult = SMWM;
BigwordMult =BGWM;
corruptMult = CORM;
spaceMult = SPM;

int size = numspec;
popsize = size;
population = new Child[popsize];
childpopulation = new Child[popsize];//Test(popsize);
newpopulation = new Child[popsize];//Test(popsize);
for (int i = 0; i < popsize; i++) {
    population[i].newcorrupt = new char[length];
    population[i].distance = new int[length];
    population[i].fitness = new double[length];
    population[i].mutation = new double[length*8];
}
for (int i = 0; i < popsize; i++) {
    childpopulation[i].newcorrupt = new char[length];
    childpopulation[i].distance = new int[length];
    childpopulation[i].fitness = new double[length];
    childpopulation[i].mutation = new double[length*8];
}
for (int i = 0; i < popsize; i++) {
    newpopulation[i].newcorrupt = new char[length];
    newpopulation[i].distance = new int[length];
    newpopulation[i].fitness = new double[length];
    newpopulation[i].mutation = new double[length*8];
}

ifstream ifile1(infile1.c_str());
ifstream ifile2(infile2.c_str());
ifstream ifile3(infile3.c_str());
ifstream ifile4(infile4.c_str());
ifstream ifile5(infile5.c_str());
ifstream ifile6(infile6.c_str());
ifstream ifile7(infile7.c_str());
ifstream ifile8(infile8.c_str());
ifstream ifile9(infile9.c_str());
ifstream ifile10(infile10.c_str());
ifstream ifile11(infile11.c_str());
ifstream ifile12(infile12.c_str());
ifstream ifile13(infile13.c_str());

```

```
ifstream ifile14(infile14.c_str());
ifstream ifile15(infile15.c_str());
ifstream ifile16(infile16.c_str());
ifstream ifile17(infile17.c_str());
ifstream ifile18(infile18.c_str());
ifstream ifile19(infile19.c_str());
ifstream ifile20(infile20.c_str());
ifstream ifile21(infile21.c_str());
ifstream ifile22(infile22.c_str());
ifstream ifile23(infile23.c_str());
```

```
quickprime[0] = 41;
quickprime[1] = 449;
quickprime[2] = 2039;
quickprime[3] = 6397;
quickprime[4] = 12973;
quickprime[5] = 23531;
quickprime[6] = 31189;
quickprime[7] = 34871;
quickprime[8] = 34871;
quickprime[9] = 31189;
quickprime[10] = 24151;
quickprime[11] = 18211;
quickprime[12] = 12973;
quickprime[13] = 6397;
quickprime[14] = 5099;
quickprime[15] = 2039;
quickprime[16] = 1493;
quickprime[17] = 797;
quickprime[18] = 337;
quickprime[19] = 79;
quickprime[20] = 41;
quickprime[21] = 11;
quickprime[22] = 7;
```

```
for(int i1 =0; i1 < 41;i1++){
    getline(ifile1,in1[i1]);
```

```
}
```

```
for(int i2 =0; i2 < 449;i2++){
    getline(ifile2,in2[i2]);
```

```
}
```

```
for(int i3 =0; i3 < 2039;i3++){
    getline(ifile3, in3[i3]);
```

```
}
```

```
for(int i4 =0; i4 < 6397;i4++){
    getline(ifile4, in4[i4]);
```

```

}

for(int i5 =0; i5 < 12973;i5++){
    getline(ifile5, in5[i5]);
}

for(int i6 =0; i6 < 23531;i6++){
    getline(ifile6,in6[i6]);
}

for(int i7 =0; i7 < 31189;i7++){
    getline(ifile7,in7[i7]);
}

for(int i8 =0; i8 < 34871;i8++){
    getline(ifile8,in8[i8]);
}

for(int i9 =0; i9 < 34871;i9++){
    getline(ifile9,in9[i9]);
    ifile9 >> in9[i9] ;
}

for(int i10 =0; i10 < 31189;i10++){
    ifile10 >> in10[i10] ;
    getline(ifile10,in10[i10]);
}

for(int i11 =0; i11 < 24151;i11++){
    ifile11 >> in11[i11] ;
    getline(ifile11,in11[i11]);
}

for(int i12 =0; i12 < 18211;i12++){
    ifile12 >> in12[i12] ;
    getline(ifile12,in12[i12]);
}

for(int i13 =0; i13 < 12973;i13++){
    ifile13 >> in13[i13] ;
    getline(ifile13,in13[i13]);
}

for(int i14 =0; i14 < 6397;i14++){
    ifile14 >> in14[i14] ;
    getline(ifile14,in14[i14]);
}

for(int i15 =0; i15 < 5099;i15++){
    ifile15 >> in15[i15] ;
    getline(ifile15,in15[i15]);
}

for(int i16 =0; i16 < 2039;i16++){
    ifile16 >> in16[i16] ;
    getline(ifile16,in16[i16]);
}

```

```

}

for(int i17 =0; i17 < 1493;i17++){
    ifile17 >> in17[i17] ;
    getline(ifile17,in17[i17]);
}

for(int i18 =0; i18 < 797;i18++){
    ifile18 >> in18[i18] ;
    getline(ifile18,in18[i18]);
}

for(int i19 =0; i19 < 337;i19++){
    ifile19 >> in19[i19] ;
    getline(ifile19,in19[i19]);
}

for(int i20 =0; i20 < 79;i20++){
    ifile20 >> in20[i20] ;
    getline(ifile20,in20[i20]);
}

for(int i21 =0; i21 < 41;i21++){
    ifile21 >> in21[i21] ;
    getline(ifile21,in21[i21]);
}

for(int i22 =0; i22 < 11;i22++){
    ifile22 >> in22[i22] ;
    getline(ifile22,in22[i22]);
}

for(int i23 =0; i23 < 7;i23++){
    ifile23 >> in23[i23] ;
    getline(ifile23,in23[i23]);
}

Randomize(size);
TestPrint1();
for(int ij=0; ij<gens; ij++){
    Breed();
    if(ij%50 ==0){
        cout << " Iteration " << ij << endl;
    }
}

}

GenAlg::~GenAlg(void)
{
    delete ortext;
    delete ortextbits;
    delete target;
    delete population;
}

```

```
}
```

```
void GenAlg::TargetEN(void)
{
    for(int i = 0; i<8*length;i++){
        if(i%8!=0){
            target[i]=percor;
        }
        else{
            target[i]= -1;
        }
    }
}
```

```
void GenAlg::Randomize(int size)
{
```

```
    target = new double[length*8];
    for(int k=0;k<length*8;k++){
        target[k]=percor;
    }
    int z;
    popsize= size;
```

```
    for(int i = 0; i< size;i++){
```

```
        for(int k = 0;k<length;k++){
            population[i].distance[k] = 0;
            population[i].fitness[k] = 0;
        }
```

```
        char c1 = 0;
```

```
        char c2 = 0;
```

```
        for(int j=0;j<length;j++){
```

```
            c1 = (char) ortext[j];
```

```
            c2 = Makemask(j);
```

```
            if((((c1 ^ c2)>96) && ((c1 ^ c2)<127))||((c1 ^ c2)==32)){
                population[i].newcorrupt[j] = (c1 ^ c2);
            }
```

```
            if((((c1 ^ c2)>64) && ((c1 ^ c2)<91))){
                population[i].newcorrupt[j] = (c1 ^ c2)+(32);
            }
```

```
            else {population[i].newcorrupt[j] = c1;}
```

```
        }
```

```
        Distance(i, population);
```

```
        TargetCH(i, population);
```

```
        Fitness(i, false, true);
```

```

    }

    Order(1,population,0,popsize-1);

}

void GenAlg::TestPrint1(void){
    ofstream Testoutfile("Testoutfile1.txt");
    for(int i=0; i<popsize; i++){
        int j = 0;
        for(j=0; j<length; j++){

            Testoutfile << population[i].newcorrupt[j];
        }
        Testoutfile << endl << "    fitnessvalue = " <<
population[i].fitnessvalue << endl << "corruption level =
"<<population[i].corruptness<< endl;
    }
}

void GenAlg::TestPrint2(void){
    ofstream Testoutfile("Testoutfile2.txt");
    for(int i=0; i<popsize; i++){
        int j = 0;
        for(j=0; j<length; j++){

            Testoutfile << population[i].newcorrupt[j] ;
        }
        Testoutfile << endl;
        for(j=0; j<length; j++){

            Testoutfile << population[i].fitness[j] << "-";
        }
        Testoutfile << endl << "    fitnessvalue = " <<
population[i].fitnessvalue << endl << "corruption level =
"<<population[i].corruptness<< endl;
    }
}

void GenAlg::Test(int size){

    for(int i=0; i<popsize; i++){
        population[i].fitnessvalue = random()%2000;
    }

    for(int j=0; j<popsize; j++){
        cout << population[j].fitnessvalue << endl;
    }

    Order(1,population,0,popsize-1);
}

```

```

for(int k=0; k<popsize; k++){
    cout << population[k].fitnessvalue << endl;
}

}

void GenAlg::Order(int numofpops, Child *pop, int popstart, int popend)
{
    if(numofpops == 1){
        int pivot;
        if(popstart<popend){
            if( ((pop[popstart].fitnessvalue >
                (pop[(popstart+popend)/2].fitnessvalue))
                &&(pop[popstart].fitnessvalue < (pop[popend].fitnessvalue)))
                ||
                ((pop[popstart].fitnessvalue <
                (pop[(popstart+popend)/2].fitnessvalue))
                &&(pop[popstart].fitnessvalue > (pop[popend].fitnessvalue)))
            ){
                pivot = popstart;
            }
            else if(
                ((pop[popend].fitnessvalue >
                pop[(popstart+popend)/2].fitnessvalue)&&(pop[popend].fitnessvalue <
                pop[popstart].fitnessvalue))
                ||
                ((pop[popend].fitnessvalue <
                pop[(popstart+popend)/2].fitnessvalue)&&(pop[popend].fitnessvalue >
                pop[popstart].fitnessvalue))
            ){
                pivot = popend;
            }
            else {
                pivot = (popstart+popend)/2;
            }
            Child temp = pop[pivot];
            pop[pivot] = pop[popend];
            pop[popend] = temp;
            pivot = popend;

            int i=popstart-1;
            int j= 0;
            for(j = popstart; j < popend; j++){
                if(pop[j].fitnessvalue >= pop[pivot].fitnessvalue){
                    i++;
                    temp = pop[j];
                    pop[j]= pop[i];
                    pop[i]= temp;
                }
            }
        }
    }
}

```

```

    }

    temp = pop[pivot];
    pop[pivot] = pop[i+1];
    pop[i+1]=temp;

    Order(1, pop,popstart,i);
    Order(1, pop,i+2,popend);
}

}

```

```

else if(numofpops == 2){
    Order(1,pop,0,popsize-1);
    Order(1,population,0,popsize-1);
    Child tempChild;

    int populationcount= 0;
    int popcount = 0;

```

```

    for(int newcount= 0;newcount < popsize;newcount++){
        if((populationcount<popsize)&&((population[populationcount].fitnessvalue >
        pop[popcount].fitnessvalue)))
            {
                while ((populationcount < popsize-1) &&
                ((population[populationcount].fitnessvalue==population[populationcount+1].fitnessvalue))){
                    populationcount++;
                }

```

```

                tempChild.newcorrupt = newpopulation[newcount].newcorrupt;
                tempChild.distance = newpopulation[newcount].distance;
                tempChild.fitness = newpopulation[newcount].fitness;
                tempChild.mutation = newpopulation[newcount].mutation;
                tempChild.fitnessvalue = newpopulation[newcount].fitnessvalue;
                tempChild.corruptness = newpopulation[newcount].corruptness;

```

```

        newpopulation[newcount].newcorrupt =
population[populationcount].newcorrupt;
        newpopulation[newcount].distance =
population[populationcount].distance;
        newpopulation[newcount].fitness =
population[populationcount].fitness;
        newpopulation[newcount].mutation =
population[populationcount].mutation;
        newpopulation[newcount].fitnessvalue =
population[populationcount].fitnessvalue;
        newpopulation[newcount].corruptness =
population[populationcount].corruptness;

        population[populationcount].newcorrupt = tempChild.newcorrupt;
        population[populationcount].distance = tempChild.distance;
        population[populationcount].fitness = tempChild.fitness;
        population[populationcount].mutation = tempChild.mutation;
        population[populationcount].fitnessvalue =
tempChild.fitnessvalue;
        population[populationcount].corruptness = tempChild.corruptness;

        populationcount++;
    }
    else
    {
        tempChild.newcorrupt = newpopulation[newcount].newcorrupt;
        tempChild.distance = newpopulation[newcount].distance;
        tempChild.fitness = newpopulation[newcount].fitness;
        tempChild.mutation = newpopulation[newcount].mutation;
        tempChild.fitnessvalue = newpopulation[newcount].fitnessvalue;
        tempChild.corruptness = newpopulation[newcount].corruptness;

        newpopulation[newcount].newcorrupt = pop[popcount].newcorrupt;
        newpopulation[newcount].distance = pop[popcount].distance;
        newpopulation[newcount].fitness = pop[popcount].fitness;
        newpopulation[newcount].mutation = pop[popcount].mutation;
        newpopulation[newcount].fitnessvalue =
pop[popcount].fitnessvalue;
        newpopulation[newcount].corruptness = pop[popcount].corruptness;

        pop[popcount].newcorrupt = tempChild.newcorrupt;
        pop[popcount].distance = tempChild.distance;
        pop[popcount].fitness = tempChild.fitness;
        pop[popcount].mutation = tempChild.mutation;
        pop[popcount].fitnessvalue = tempChild.fitnessvalue;
        pop[popcount].corruptness = tempChild.corruptness;

        popcount++;
    }
}

/* *****INTERBREED***** */
for(int j=0; j<popsize/2; j=j+2){
    tempChild.newcorrupt = newpopulation[j].newcorrupt;
    tempChild.distance = newpopulation[j].distance;
    tempChild.fitness = newpopulation[j].fitness;

```

```

tempChild.mutation = newpopulation[j].mutation;
tempChild.fitnessvalue = newpopulation[j].fitnessvalue;
tempChild.corruptness = newpopulation[j].corruptness;

newpopulation[j].newcorrupt = population[j].newcorrupt;
newpopulation[j].distance = population[j].distance;
newpopulation[j].fitness = population[j].fitness;
newpopulation[j].mutation = population[j].mutation;
newpopulation[j].fitnessvalue = population[j].fitnessvalue;
newpopulation[j].corruptness = population[j].corruptness;

population[j].newcorrupt = tempChild.newcorrupt;
population[j].distance = tempChild.distance;
population[j].fitness = tempChild.fitness;
population[j].mutation = tempChild.mutation;
population[j].fitnessvalue = tempChild.fitnessvalue;
population[j].corruptness = tempChild.corruptness;

tempChild.newcorrupt = newpopulation[(popsize/2)-(j+1)].newcorrupt;
tempChild.distance = newpopulation[(popsize/2)-(j+1)].distance;
tempChild.fitness = newpopulation[(popsize/2)-(j+1)].fitness;
tempChild.mutation = newpopulation[(popsize/2)-(j+1)].mutation;
tempChild.fitnessvalue =
newpopulation[(popsize/2)-(j+1)].fitnessvalue;
tempChild.corruptness =
newpopulation[(popsize/2)-(j+1)].corruptness;

newpopulation[(popsize/2)-(j+1)].newcorrupt =
population[j+1].newcorrupt;
newpopulation[(popsize/2)-(j+1)].distance =
population[j+1].distance;
newpopulation[(popsize/2)-(j+1)].fitness = population[j+1].fitness;
newpopulation[(popsize/2)-(j+1)].mutation =
population[j+1].mutation;
newpopulation[(popsize/2)-(j+1)].fitnessvalue =
population[j+1].fitnessvalue;
newpopulation[(popsize/2)-(j+1)].corruptness =
population[j+1].corruptness;

population[j+1].newcorrupt = tempChild.newcorrupt;
population[j+1].distance = tempChild.distance;
population[j+1].fitness = tempChild.fitness;
population[j+1].mutation = tempChild.mutation;
population[j+1].fitnessvalue = tempChild.fitnessvalue;
population[j+1].corruptness = tempChild.corruptness;

}

for (int j = popsize/2; j < popsize; j++) {
*/
for (int j = 0; j < popsize; j++) {
tempChild.newcorrupt = newpopulation[j].newcorrupt;
tempChild.distance = newpopulation[j].distance;
tempChild.fitness = newpopulation[j].fitness;
tempChild.mutation = newpopulation[j].mutation;

```

```

tempChild.fitnessvalue = newpopulation[j].fitnessvalue;
tempChild.corruptness = newpopulation[j].corruptness;

newpopulation[j].newcorrupt = population[j].newcorrupt;
newpopulation[j].distance = population[j].distance;
newpopulation[j].fitness = population[j].fitness;
newpopulation[j].mutation = population[j].mutation;
newpopulation[j].fitnessvalue = population[j].fitnessvalue;
newpopulation[j].corruptness = population[j].corruptness;

population[j].newcorrupt = tempChild.newcorrupt;
population[j].distance = tempChild.distance;
population[j].fitness = tempChild.fitness;
population[j].mutation = tempChild.mutation;
population[j].fitnessvalue = tempChild.fitnessvalue;
population[j].corruptness = tempChild.corruptness;

    }
}
}

void GenAlg::Breed(void)
{
    for(int j=0; j<length; j++){
        cout << population[0].newcorrupt[j];
    }
    cout << endl;
    int rand;

    for(int i = 0; i < popsize; i=i+2){
        rand = random() % length;
        for(int j = 0; j < rand; j++){
            childpopulation[i].newcorrupt[j] = population[i].newcorrupt[j];
            childpopulation[i].distance[j] = population[i].distance[j];
            childpopulation[i].fitness[j] = population[i].fitness[j];
            for(int m=0; m<8; m++){
                childpopulation[i].mutation[8*j+m] =
population[i].mutation[8*j+m];
            }
            childpopulation[i+1].newcorrupt[j] = population[i+1].newcorrupt[j];
            childpopulation[i+1].distance[j] = population[i+1].distance[j];
            childpopulation[i+1].fitness[j] = population[i+1].fitness[j];
            for(int m=0; m<8; m++){
                childpopulation[i+1].mutation[8*j+m] =
population[i+1].mutation[8*j+m];
            }
        }
    }
    for(int j = rand; j < length; j++){
        childpopulation[i].newcorrupt[j] = population[i+1].newcorrupt[j];
        childpopulation[i].distance[j] = population[i+1].distance[j];
        childpopulation[i].fitness[j] = population[i+1].fitness[j];
    }
}

```

```

        for(int m=0; m<8; m++){
            childpopulation[i].mutation[8*j+m] =
population[i+1].mutation[8*j+m];
        }

        childpopulation[i+1].newcorrupt[j] = population[i].newcorrupt[j];
        childpopulation[i+1].distance[j] = population[i].distance[j];
        childpopulation[i+1].fitness[j] = population[i].fitness[j];
        for(int m=0; m<8; m++){
            childpopulation[i+1].mutation[8*j+m] =
population[i].mutation[8*j+m];
        }
    }

    if(true){
        Mutate(i, childpopulation);
        Mutate(i+1, childpopulation);
    }
    Distance(i, childpopulation);
    Distance(i+1, childpopulation);

    TargetCH(i, childpopulation);
    TargetCH(i+1, childpopulation);
    Fitness(i, true, false);
    Fitness(i+1, true, false);
}
ofstream outfile("childfile.txt");
for(int i=0; i<10; i++){
    int j = 0;
    for(j=0; j<length; j++){
        outfile << childpopulation[i].newcorrupt[j];
    }
    outfile << endl << "fitnessvalue = " << population[i].fitnessvalue <<
endl << "corruption level = " << population[i].corruptness << endl;
}
    Order(2, childpopulation,0,popsize);
}

void GenAlg::Mutate(int inChild, Child *pop)
{
    int randVal;
    char maskVal=0;
    randVal = random() % (8*(length));
    int charNum = randVal / 8;
    int bitNum = randVal % 8;

    if (charNum >= length)
        cout << "ERROR IN MUTATE: character out of range" << endl;
    if (bitNum >= 8)
        cout << "ERROR IN MUTATE: bad bit selected" << endl;
    if (randVal < 0)
        cout << "ERROR IN MUTATE: negative random value" << endl;
    if (inChild >= popsize)
        cout << "ERROR IN MUTATE: negative random value" << endl;
    cout << flush;
}

```

```

char c1 = '\0';
char c2 = '\0';
c1 = (char) pop[inChild].newcorrupt[charNum];
c2 = Makemask(charNum);
// cout << (int)(c1 ^ c2) << endl;
if((((c1 ^ c2)>96) && ((c1 ^ c2)<123))|((c1 ^ c2)==32)){
    pop[inChild].newcorrupt[charNum] = (c1 ^ c2);
//    cout << "dup " << (int)(c1 ^ c2) << endl;

}
}

void GenAlg::Distance(int inChild, Child *pop)
{

    int corrupcount = 0;
    for(int i = 0; i < length; i++){

        pop[inChild].distance[i] = Iscloseto((char)pop[inChild].newcorrupt[i],
(char)ortext[i]);
        corrupcount = corrupcount+pop[inChild].distance[i] ;
    }
    pop[inChild].corruptness = (100*(double)corrupcount/(length*8));

}

void GenAlg::TargetCH(int inChild, Child *pop)
{
    for(int i=0;i<8*length;i++){
        pop[inChild].mutation[i]=percor ;
    }
}

void GenAlg::Fitness(int inChild, bool Cpop, bool Opop)
{

    string temp = "";
    int count;
    //int spaces = 0;
    int i = 0;
    int j = 0;

    double fit=0;
    double dist=0;
    double abvalcor=0;

    if(Cpop){
        for(i=0; i< length; i++){
            if(childpopulation[inChild].newcorrupt[i] == ' '){
                childpopulation[inChild].fitness[i]=spaceMult;
                if((i<length-1)&&(childpopulation[inChild].newcorrupt[i+1] == '
'))
                    {childpopulation[inChild].fitness[i]=0;}
                //spaces++;
            }
        }
    }
}

```

```

        else childpopulation[inChild].fitness[i]=0;
    }
    for(i=0; i < length; i++){
        if(i==0){
            count = 0;
            temp = "";
            for(j=i; (childpopulation[inChild].newcorrupt[j] != ' ' && j
<length); j++){
                count++;
                temp = temp + childpopulation[inChild].newcorrupt[j];
            }

            if(Dictionary(count,temp)){
                for(int k=i;k < j; k++){
                    if (count <=3){
                        childpopulation[inChild].fitness[k] = SmallwordMult;
                    }
                    else if(count >3){
                        childpopulation[inChild].fitness[k] =
BigwordMult;/*(count/2);
                    }
                }
            }
            if(j>0)
                i=j-1;
        }
        else

            if(childpopulation[inChild].newcorrupt[i]==' '){
                count = 0;
                temp = "";
                if(i<length-1){
                    for(j=i+1; childpopulation[inChild].newcorrupt[j]!=' '&
j <length;j++){
                        count++;
                        temp = temp +
childpopulation[inChild].newcorrupt[j];
                    }

                    if(Dictionary(count,temp)){

                        for(int k=i; k < j; k++){
                            if (count <=3){
                                childpopulation[inChild].fitness[k] =
SmallwordMult;
                            }
                            else if(count >3){
                                childpopulation[inChild].fitness[k] =
BigwordMult;/*(count/2);
                            }
                        }
                    }
                    i = j - 1;
                }
            }
        }
    }
}

```

```

for(int m=0; m< length; m++){
    fit = fit+childpopulation[inChild].fitness[m];
}

for(int m=0; m< length; m++){
    dist = dist+childpopulation[inChild].distance[m];
}
abvalcor = percor*100 - dist;
if(abvalcor<0){
    abvalcor=abvalcor*(-1);
}
childpopulation[inChild].fitnessvalue = fit-(corruptMult*abvalcor);
}

if(OPop){
for(i=0; i< length; i++){
    if(population[inChild].newcorrupt[i] == ' '){
        population[inChild].fitness[i]=spaceMult;
        if((i<length-1)&&(childpopulation[inChild].newcorrupt[i+1] == '
'))
            {childpopulation[inChild].fitness[i]=0;}
            // spaces++;
        }
        else population[inChild].fitness[i]=0;
    }

for(i=0; i < length; i++){
    if(i==0){
        count = 0;
        temp = "";
        for(j=i; population[inChild].newcorrupt[j]!=' '&& j
<length;j++){
            count++;
            temp = temp + population[inChild].newcorrupt[j];
        }

        if(Dictionary(count,temp)){

            for(int k=i; k < j;k++){
                if (count <=3){
                    population[inChild].fitness[k] = SmallwordMult;
                }
                else if(count >3){
                    population[inChild].fitness[k] =
BigwordMult;/*(count/2);
                }
            }
        }
        if(j>0)
            i = j - 1;
    }
}

```

```

else
    if(population[inChild].newcorrupt[i]==' '){
        count = 0;
        temp = "";
        if(i<length){
            for(j=i+1; population[inChild].newcorrupt[j]!=' '&& j
<length;j++){
                count++;
                temp = temp + population[inChild].newcorrupt[j];
            }

            if(Dictionary(count,temp)){

                for(int k=i; k < j;k++){
                    if (count <=3){
                        population[inChild].fitness[k] =
SmallwordMult;
                    }
                    else if(count >3){
                        population[inChild].fitness[k] =
BigwordMult;/*(count/2);
                    }
                }
                i = j - 1;
            }
        }
    }
    for(int m=0; m< length; m++){
        fit = fit+population[inChild].fitness[m];
    }

    for(int m=0; m< length; m++){
        dist = dist+population[inChild].distance[m];
    }
    abvalcor = percor*100 - dist;
    if(abvalcor<0){
        abvalcor=abvalcor*(-1);
    }
    population[inChild].fitnessvalue = fit-(corruptMult*abvalcor);
}
}

```

```

bool GenAlg::Dictionary(int count, string word)
{
    int h =0;

    if(count > 23||count<1){
        return false;
    }
    else {

```

```

for(int i=0; i<count;i++){
    h = (h * 257) + (char) word[i];
    h = h % quickprime[count-1];
}
}

if(count == 1){
    if (in1[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in1[h] != ""){

            if (in1[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 2){
    if (in2[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in2[h] != ""){
            if (in2[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 3){
    if (in3[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in3[h] != ""){
            if (in3[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 4){
    if (in4[h] == word){
        return true;
    }
}

```

```

    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in4[h] != ""){
            if (in4[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 5){
    if (in5[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in5[h] != ""){
            if (in5[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 6){
    if (in6[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in6[h] != ""){
            if (in6[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 7){
    if (in7[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in7[h] != ""){
            if (in7[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}
}

```

```

if(count == 8){
  if (in8[h] == word){
    return true;
  }
  else {
    h = (h + 1) % quickprime[count-1];
    while(in8[h] != ""){

      if (in8[h] == word)
        return true;
      h = (h + 1) % quickprime[count-1];
    }
  }
  return false;
}

```

```

if(count == 9){
  if (in9[h] == word){
    return true;
  }
  else {
    h = (h + 1) % quickprime[count-1];
    while(in9[h] != ""){
      if (in9[h] == word)
        return true;
      h = (h + 1) % quickprime[count-1];
    }
  }
  return false;
}

```

```

if(count == 10){

  if (in10[h] == word){

    return true;
  }
  else {
    h = (h + 1) % quickprime[count-1];
    while(in10[h] != ""){
      if (in10[h] == word)
        return true;
      h = (h + 1) % quickprime[count-1];
    }
  }
  return false;
}

```

```

if(count == 11){
  if (in11[h] == word){
    return true;
  }
  else {
    h = (h + 1) % quickprime[count-1];

```

```

        while(in11[h] != ""){
            if (in11[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 12){
    if (in12[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in12[h] != ""){

            if (in12[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 13){
    if (in13[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in13[h] != ""){

            if (in13[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }

    return false;
}

if(count == 14){
    if (in14[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in14[h] != ""){

            if (in14[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}
}

```

```

if(count == 15){
  if (in15[h] == word){
    return true;
  }
  else {
    h = (h + 1) % quickprime[count-1];
    while(in15[h] != ""){

      if (in15[h] == word)
        return true;
      h = (h + 1) % quickprime[count-1];
    }
  }
  return false;
}

```

```

if(count == 16){
  if (in16[h] == word){
    return true;
  }
  else {
    h = (h + 1) % quickprime[count-1];
    while(in16[h] != ""){
      if (in16[h] == word)
        return true;
      h = (h + 1) % quickprime[count-1];
    }
  }
  return false;
}

```

```

if(count == 17){
  if (in17[h] == word){
    return true;
  }
  else {
    h = (h + 1) % quickprime[count-1];
    while(in17[h] != ""){

      if (in17[h] == word)
        return true;

      h = (h + 1) % quickprime[count-1];
    }
  }
  return false;
}

```

```

if(count == 18){
  if (in18[h] == word){
    return true;
  }
  else {
    h = (h + 1) % quickprime[count-1];
    while(in18[h] != ""){

```

```

        if (in18[h] == word)
            return true;
        h = (h + 1) % quickprime[count-1];
    }
}
return false;
}

if(count == 19){
    if (in19[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in19[h] != ""){

            if (in19[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 20){
    if (in20[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in20[h] != ""){

            if (in20[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 21){
    if (in21[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in21[h] != ""){

            if (in21[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}
}

```

```

if(count == 22){
    if (in22[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in22[h] != ""){
            if (in22[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}

if(count == 23){
    if (in23[h] == word){
        return true;
    }
    else {
        h = (h + 1) % quickprime[count-1];
        while(in23[h] != ""){
            if (in23[h] == word)
                return true;
            h = (h + 1) % quickprime[count-1];
        }
    }
    return false;
}
}

```

```

char GenAlg::Makemask(int place)

```

```

{
    char randomval = '\0';
    int i;
    double j;
    for(int count = 1; count<8; count++){
        i = random() % 100;
        j = (double)i/100;

        if(j < target[place*8 + count]){
            (unsigned)randomval = (unsigned)randomval * 2;
            (unsigned)randomval = (unsigned)randomval + 1;
        }
        else{
            (unsigned)randomval = (unsigned)randomval << 1;
        }
    }
}

```

```
    return randomval;
}
```

```
int Iscloseto(char inchar1, char inchar2)
```

```
{
    int count = 0;

    if((inchar1/128)!=inchar2/128){count++;}
    inchar1=inchar1%128;inchar2=inchar2%128;
    if((inchar1/64)!=inchar2/64){count++;}
    inchar1=inchar1%64;inchar2=inchar2%64;
    if((inchar1/32)!=inchar2/32){count++;}
    inchar1=inchar1%32;inchar2=inchar2%32;
    if((inchar1/16)!=inchar2/16){count++;}
    inchar1=inchar1%16;inchar2=inchar2%16;
    if((inchar1/8)!=inchar2/8){count++;}
    inchar1=inchar1%8;inchar2=inchar2%8;
    if((inchar1/4)!=inchar2/4){count++;}
    inchar1=inchar1%4;inchar2=inchar2%4;
    if((inchar1/2)!=inchar2/2){count++;}
    inchar1=inchar1%2;inchar2=inchar2%2;
    if((inchar1/1)!=inchar2/1){count++;}

    return count;
}
```

Appendix D

Dictionary/Hash Tables Implementation

```
*****Dictionary.h*****

#ifndef DICTIONARY_H
#define DICTIONARY_H
#include <string>
using namespace std;

class Dictionary {
public:

    Dictionary(void);

    ~Dictionary(void);

    void Setinfile(void);

    void CountLetterLength(void);

    void PrintLetterLength(void);

    void IntializeOutArrays(void);
    void HashList(void);
    void PrintOutArrays(void); //Print Dictionary files

private:

    char infile[25];
    int letterlength[99];

    int quickprime[23];

    string out1[41];
    string out2[449];
    string out3[2039];
    string out4[6397];
    string out5[12973];
    string out6[23531];
    string out7[31189];
    string out8[34871];
    string out9[34871];
    string out10[31189];
    string out11[24151];
```

```

string out12[18211];
string out13[12973];
string out14[6397];
string out15[5099];
string out16[2039];
string out17[1493];
string out18[797];
string out19[337];
string out20[79];
string out21[41];
string out22[11];
string out23[7];

};

#endif

*****Dictionary.cc*****

#include <string>
#include <fstream>
#include <ostream>
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <cmath>
#include "Dictionary.h"

const string outfile1 = "HashDictionary1.txt";
const string outfile2 = "HashDictionary2.txt";
const string outfile3 = "HashDictionary3.txt";
const string outfile4 = "HashDictionary4.txt";
const string outfile5 = "HashDictionary5.txt";
const string outfile6 = "HashDictionary6.txt";
const string outfile7 = "HashDictionary7.txt";
const string outfile8 = "HashDictionary8.txt";
const string outfile9 = "HashDictionary9.txt";
const string outfile10 = "HashDictionary10.txt";
const string outfile11 = "HashDictionary11.txt";
const string outfile12 = "HashDictionary12.txt";
const string outfile13 = "HashDictionary13.txt";
const string outfile14 = "HashDictionary14.txt";
const string outfile15 = "HashDictionary15.txt";
const string outfile16 = "HashDictionary16.txt";
const string outfile17 = "HashDictionary17.txt";
const string outfile18 = "HashDictionary18.txt";
const string outfile19 = "HashDictionary19.txt";

```

```

const string outfile20 = "HashDictionary20.txt";
const string outfile21 = "HashDictionary21.txt";
const string outfile22 = "HashDictionary22.txt";
const string outfile23 = "HashDictionary23.txt";

```

```

using namespace std;

```

```

Dictionary::Dictionary(void)

```

```

{
    for (int j=0; j<24;j++){
        letterlength[j]=0;
    }
}

```

```

Dictionary::~Dictionary(void)

```

```

{
    delete infile;
}

```

```

void Dictionary::Setinfile(void)

```

```

{
    cout << "Please enter the name of the Dictionary file. >> ";
    cin >> infile;
}

```

```

void Dictionary::CountLetterLength(void)

```

```

{
    string c1;           //read in value
    ifstream file(infile);

    while(file>>c1){
        letterlength[c1.length()]=letterlength[c1.length()+1];
    }
}

```

```

void Dictionary::PrintLetterLength(void)

```

```

{
    for(int count = 0; count<99; count++){
        cout << "There are " << letterlength[count] << " for enteries of lenght
" << count << endl;
    }
}

```

```

void Dictionary::IntializeOutArrays(void)

```

```

{
    for(int i1 =0; i1 < 41;i1++){

```

```

    out1[i1]= "";
}

for(int i2 =0; i2 < 449;i2++){
    out2[i2]= "";
}

for(int i3 =0; i3 < 2039;i3++){
    out3[i3]= "";
}

for(int i4 =0; i4 < 6397;i4++){
    out4[i4]= "";
}

for(int i5 =0; i5 < 12973;i5++){
    out5[i5]= "";
}

for(int i6 =0; i6 < 23531;i6++){
    out6[i6]= "";
}

for(int i7 =0; i7 < 31189;i7++){
    out7[i7]= "";
}

for(int i8 =0; i8 < 34871;i8++){
    out8[i8]= "";
}

for(int i9 =0; i9 < 34871;i9++){
    out9[i9]= "";
}

for(int i10 =0; i10 < 31189;i10++){
    out10[i10]= "";
}

for(int i11 =0; i11 < 24151;i11++){
    out11[i11]= "";
}

for(int i12 =0; i12 < 18211;i12++){
    out12[i12]= "";
}

for(int i13 =0; i13 < 12973;i13++){
    out13[i13]= "";
}

for(int i14 =0; i14 < 6397;i14++){
    out14[i14]= "";
}

for(int i15 =0; i15 < 5099;i15++){

```

```

    out15[i15]= "";
}

for(int i16 =0; i16 < 2039;i16++){
    out16[i16]= "";
}

for(int i17 =0; i17 < 1493;i17++){
    out17[i17]= "";
}

for(int i18 =0; i18 < 797;i18++){
    out18[i18]= "";
}

for(int i19 =0; i19 < 337;i19++){
    out19[i19]= "";
}

for(int i20 =0; i20 < 79;i20++){
    out20[i20]= "";
}

for(int i21 =0; i21 < 41;i21++){
    out21[i21]= "";
}

for(int i22 =0; i22 < 11;i22++){
    out22[i22]= "";
}

for(int i23 =0; i23 < 7;i23++){
    out23[i23]= "";
}

}

void Dictionary::HashList(void)
{
    string c1;          //read in value
    ifstream file(infile);

    quickprime[0] = 41;
    quickprime[1] = 449;
    quickprime[2] = 2039;
    quickprime[3] = 6397;
    quickprime[4] = 12973;
    quickprime[5] = 23531;
    quickprime[6] = 31189;
    quickprime[7] = 34871;
    quickprime[8] = 34871;
    quickprime[9] = 31189;
    quickprime[10] = 24151;
    quickprime[11] = 18211;
}

```

```

quickprime[12] = 12973;
quickprime[13] = 6397;
quickprime[14] = 5099;
quickprime[15] = 2039;
quickprime[16] = 1493;
quickprime[17] = 797;
quickprime[18] = 337;
quickprime[19] = 79;
quickprime[20] = 41;
quickprime[21] = 11;
quickprime[22] = 7;

int h;

while(file>>c1){
    h = 0;

    //cout << c1 << endl;

    int length = c1.length();

    if(length > 23){
        cout << c1 << endl;
    }
    else {
        for(int i=0; i<length;i++){
            h = (h * 257) + (char) c1[i];
            h = h % quickprime[length-1];
        }
    }

    if(length == 1){
        cout << "Got here!!" << endl;
        if (out1[h] == ""){
            out1[h] = c1;
        }
        else {
            h++;
            while(out1[h] != ""){
                h++;
                if (h == quickprime[length-1])
                    h=0;
            }
            out1[h] = c1;
        }
    }

    if(length == 2){
        if (out2[h] == ""){
            out2[h] = c1;
        }
        else {
            h++;
            while(out2[h] != ""){
                h++;
                if (h == quickprime[length-1])

```

```

        h=0;
    }
    out2[h] = c1;
}

if(length == 3){
    if (out3[h] == ""){
        out3[h] = c1;
    }
    else {
        h++;
        while(out3[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out3[h] = c1;
    }
}

if(length == 4){
    if (out4[h] == ""){
        out4[h] = c1;
    }
    else {
        h++;
        while(out4[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out4[h] = c1;
    }
}

if(length == 5){
    if (out5[h] == ""){
        out5[h] = c1;
    }
    else {
        h++;
        while(out5[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out5[h] = c1;
    }
}

if(length == 6){
    if (out6[h] == ""){
        out6[h] = c1;
    }
    else {

```

```

        h++;
        while(out6[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out6[h] = c1;
    }
}

if(length == 7){
    if (out7[h] == ""){
        out7[h] = c1;
    }
    else {
        h++;
        while(out7[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out7[h] = c1;
    }
}

if(length == 8){
    if (out8[h] == ""){
        out8[h] = c1;
    }
    else {
        h++;
        while(out8[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out8[h] = c1;
    }
}

if(length == 9){
    if (out9[h] == ""){
        out9[h] = c1;
    }
    else {
        h++;
        while(out9[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out9[h] = c1;
    }
}

if(length == 10){

```

```

if (out10[h] == ""){
    out10[h] = c1;
}
else {
    h++;
    while(out10[h] != ""){
        h++;
        if (h == quickprime[length-1])
            h=0;
    }
    out10[h] = c1;
}
}

```

```

if(length == 11){
    if (out11[h] == ""){
        out11[h] = c1;
    }
    else {
        h++;
        while(out11[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out11[h] = c1;
    }
}
}

```

```

if(length == 12){
    if (out12[h] == ""){
        out12[h] = c1;
    }
    else {
        h++;
        while(out12[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out12[h] = c1;
    }
}
}

```

```

if(length == 13){
    if (out13[h] == ""){
        out13[h] = c1;
    }
    else {
        h++;
        while(out13[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out13[h] = c1;
    }
}
}

```

```

    }
}

if(length == 14){
    if (out14[h] == ""){
        out14[h] = c1;
    }
    else {
        h++;
        while(out14[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out14[h] = c1;
    }
}

if(length == 15){
    if (out15[h] == ""){
        out15[h] = c1;
    }
    else {
        h++;
        while(out15[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out15[h] = c1;
    }
}

if(length == 16){
    if (out16[h] == ""){
        out16[h] = c1;
    }
    else {
        h++;
        while(out16[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out16[h] = c1;
    }
}

if(length == 17){
    if (out17[h] == ""){
        out17[h] = c1;
    }
    else {
        h++;
        while(out17[h] != ""){
            h++;

```

```

        if (h == quickprime[length-1])
            h=0;
    }
    out17[h] = c1;
}

if(length == 18){
    if (out18[h] == ""){
        out18[h] = c1;
    }
    else {
        h++;
        while(out18[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out18[h] = c1;
    }
}

if(length == 19){
    if (out19[h] == ""){
        out19[h] = c1;
    }
    else {
        h++;
        while(out19[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out19[h] = c1;
    }
}

if(length == 20){
    if (out20[h] == ""){
        out20[h] = c1;
    }
    else {
        h++;
        while(out20[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out20[h] = c1;
    }
}

if(length == 21){
    if (out21[h] == ""){
        out21[h] = c1;
    }
    else {
        h++;
    }
}

```

```

        while(out21[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out21[h] = c1;
    }
}

if(length == 22){
    if (out22[h] == ""){
        out22[h] = c1;
    }
    else {
        h++;
        while(out22[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out22[h] = c1;
    }
}

if(length == 23){
    if (out23[h] == ""){
        out23[h] = c1;
    }
    else {
        h++;
        while(out23[h] != ""){
            h++;
            if (h == quickprime[length-1])
                h=0;
        }
        out23[h] = c1;
    }
}

}
}

```

```

void Dictionary::PrintOutArrays(void)
{
    ofstream outfile1(outfile1.c_str());
    ofstream outfile2(outfile2.c_str());
    ofstream outfile3(outfile3.c_str());
    ofstream outfile4(outfile4.c_str());
    ofstream outfile5(outfile5.c_str());
    ofstream outfile6(outfile6.c_str());
    ofstream outfile7(outfile7.c_str());
    ofstream outfile8(outfile8.c_str());
}

```

```
ofstream ofile9(outfile9.c_str());
ofstream ofile10(outfile10.c_str());
ofstream ofile11(outfile11.c_str());
ofstream ofile12(outfile12.c_str());
ofstream ofile13(outfile13.c_str());
ofstream ofile14(outfile14.c_str());
ofstream ofile15(outfile15.c_str());
ofstream ofile16(outfile16.c_str());
ofstream ofile17(outfile17.c_str());
ofstream ofile18(outfile18.c_str());
ofstream ofile19(outfile19.c_str());
ofstream ofile20(outfile20.c_str());
ofstream ofile21(outfile21.c_str());
ofstream ofile22(outfile22.c_str());
ofstream ofile23(outfile23.c_str());
```

```
for(int i1 =0; i1 < 41;i1++){
    ofile1 << out1[i1] << endl;
    cout << out1[i1] << endl;
}
```

```
for(int i2 =0; i2 < 449;i2++){
    ofile2 << out2[i2] << endl;
    cout << out2[i2] << endl;
}
```

```
for(int i3 =0; i3 < 2039;i3++){
    ofile3 << out3[i3] << endl;
}
```

```
for(int i4 =0; i4 < 6397;i4++){
    ofile4 << out4[i4] << endl;
}
```

```
for(int i5 =0; i5 < 12973;i5++){
    ofile5 << out5[i5] << endl;
}
```

```
for(int i6 =0; i6 < 23531;i6++){
    ofile6 << out6[i6] << endl;
}
```

```
for(int i7 =0; i7 < 31189;i7++){
    ofile7 << out7[i7] << endl;
}
```

```
for(int i8 =0; i8 < 34871;i8++){
    ofile8 << out8[i8] << endl;
}
```

```
for(int i9 =0; i9 < 34871;i9++){
    ofile9 << out9[i9] << endl;
}
```

```
for(int i10 =0; i10 < 31189;i10++){
    ofile10 << out10[i10] << endl;
}
```

```
}  
  
for(int i11 =0; i11 < 24151;i11++){  
    ofile11 << out11[i11] << endl;  
}  
  
for(int i12 =0; i12 < 18211;i12++){  
    ofile12 << out12[i12] << endl;  
}  
  
for(int i13 =0; i13 < 12973;i13++){  
    ofile13 << out13[i13] << endl;  
}  
  
for(int i14 =0; i14 < 6397;i14++){  
    ofile14 << out14[i14] << endl;  
}  
  
for(int i15 =0; i15 < 5099;i15++){  
    ofile15 << out15[i15] << endl;  
}  
  
for(int i16 =0; i16 < 2039;i16++){  
    ofile16 << out16[i16] << endl;  
}  
  
for(int i17 =0; i17 < 1493;i17++){  
    ofile17 << out17[i17] << endl;  
}  
  
for(int i18 =0; i18 < 797;i18++){  
    ofile18 << out18[i18] << endl;  
}  
  
for(int i19 =0; i19 < 337;i19++){  
    ofile19 << out19[i19] << endl;  
}  
  
for(int i20 =0; i20 < 79;i20++){  
    ofile20 << out20[i20] << endl;  
}  
  
for(int i21 =0; i21 < 41;i21++){  
    ofile21 << out21[i21] << endl;  
}  
  
for(int i22 =0; i22 < 11;i22++){  
    ofile22 << out22[i22] << endl;  
}  
  
for(int i23 =0; i23 < 7;i23++){  
    ofile23 << out23[i23] << endl;  
}  
  
}
```

Appendix E

Example Run of Genetic Algorithm

The following output shows the highest fitness value specimen of a population of 2,500 at each generation for 1000 generations.

The original sentence that was used was:

```
hello my name is not a dictionary word
```

After randomly corrupting approximately 15% of the bits, and preprocessing to switch the top order bits back from one to zero, the input specimen to the Genetic Algorithm was:

```
hqlmC"ay ~!Mg!hq(ngv i"Dqq4)&nirx 7kRf
```

The following shows the top fitness specimen for each generation for 1000 generations:

The number, three lines down (0.128205) refer to the actual corruption of the specimen after the top order bits have been reversed.

The final specimen is

```
hales as name ha not a day a nary word
```

```
gemin{i{45}% driver  
entering first phase.  
0.128205  
hqlmr"ay ~iMg!hr(ngv i"qq4)fnirx 7kvf  
Iteration 0  
hqlmc"ay ~!mc!hq ngv i"!qq4)&ninx 7kbf  
hqlmj" y ~!eg!hq(ngs i Dqq4)&nirx 7kwf  
hqlmj" y ~!eg!hq(ngs i Dqq4)&nirx 7kqf
```

hqlmc"ax ~!ig! q(ngv i"lqq)&nira 7kRf
hq mb"ay ~qMgahq(ngv i" qq4)&lirx 7krf
hqlmC"ay ~!eg! q(n v i dqq4)&nirx 7kzf
hulmg yy ~!Mg!hq ngv i"dqq4 &nirx 7crf
hqlmc ax ~!ig! q(ngv i"lqq)&nhrx 7krf
hqlmC"ay ~!eg! q(n v i Dqq)&nirx 7krf
hqlmC"ay ~!eg! q(n v i Daq)&nirx 7krf
hqlmc ac ~!mw!hq ngv i dqq)&nirx 7krf
hq mb ay ~!Mg hq(ogv i dqq)&nirx 7krf
hq mb ay ~!Mg hq(ogv i dqq)&nirx 7krf
hq mb ay ~ Mg hq(ogv i dqq)&nirx 7krf
hq mb ay ~ Mg hq(ogv i dqq)&nirx 7krf
hq mb ay ~!Mg hq ogv i dqq)&nirx 7krf
hq mb ay ~ Mg hq(ogv i dqq)&nirx 7krf
hq mb ay ~ Mg hq ogv i dqq)&nirx 7krf
hq mb ay ~ Mg hq ogv i dqq)&nirx 7krf
qlmC" y ~ Mg hq ngv i Dqq)&nirx 7krf
qlmb ay ~ Mg hq ngv i Dqq)&nirx 7krf
q mC" y ~ Mg hq ngv i dqq)&nirx 7krf
q mC" y ~ Mg hq ngv i dqq)&nirx 7krf
q mC" y ~ Mg hq ngv i dqq)&nirx 7krf
ha mb ay ~ Mg hq ogv i dqq)&nirx 7krf
ha mb ay ~ Mg hq ogv i Dqq) nhbx 7kRf
ha mb ay ~ Mg hq ogv i Dqq)&nirx 7krf
ha mb ay ~ Mg hq ogv i Dqq)&nirx 7krf
ha mb ay ~ Mg hq ogv i dqq)&nirx 7kRf
ha mb ay ~ Mg hy ngv i D q)&nirx 7kRf
ha mC" y ~ Mg iq ngv i Dqq)&nirx 7krf
ha mb ay ~ Mg hq ngv a dqq)&nirx 7kRf
ha mb ay ~ Mg hq ngv a dqq)&nirx 7kRf
qlmC as ~!Mg ha ngv i dqq)&nirx 7krf
qlmC as ~!Mg ha ngv a dqq)&nirx 7kRf
qlmC as ~!Mg ha ngv a dqq)&nirx 7kRf
qlmC as ~!Mg ha ngv a dqq) nirx 7kRf
ha mc by ~ Mg ha ngv i fqq)&nirx 7krf
ha mC as ~!Me ha ngv a dqq)&nirx 7kRf
ha mC as ~ Mg ha ngv a dqq)&nirx 7kRf
ha mC as ~ Mg ha ngv a dqq)&nirx 7kRf
ha mC as ~ Mg ha ngv a dqq)&nirx 7kRf
ha mC as ~ Mg ha ngv a dqq)&nirx 7kRf
ha mC as ~ Mg ha ngv a dqq)&nirx 7kRf
ha mc by ~!Mg ha ngv a day)&nirx 7kRf
ha mc by ~!Mg ha ngv a day)&nirx 7kRf
ha mC by ~ Mg ha ngv a day)&nirx 7kRf
ha mC by ~ Mg ha ngv a day)&nirx 7kRf
ha mC by ~ Mg ha ngv a day)&nirx 7kRf
ha mC by ~ Mg ha ngv a day)&nirx 7kRf
Iteration 50
ha mC as ~ Mg ha ngv a day)&nirx 7kRf
ha mC by ~ Mg ha ngv a day)&nirx 7kRf
ha mC by ~ Mg ha ngv a day)&nirx 7kRf
ha mC as ~ me ha ngv a day)&nkrx 7kRf
ha mC as ~ me ha ngv a day)&nirx 7kRf

hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales as ~ me ha nor a day a nary 7kRf
hales as ~ me ha nor a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales by ~ me ha now a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales as ~ me ha nor a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales as ~ me ha nor a day a nary 7kRf
hales my ~ me ha not a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf

Iteration 150

hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales by ~ me ha now a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales as ~ me ha not a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales my ~ me ha not a day a nary 7kRf

hales my ~ me ha now a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales as ~ me ha not a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
halls as ~ me ha not a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales as ~ me ha not a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
halos as ~ me ha nor a day a nary 7kRf
hales my ~ me ha not a day a nary 7kRf
hales my ~ me ha not a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales as ~ me ha nor a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales as ~ me ha not a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales ex ~ me ha now a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
halls as ~ me ha now a day a nary 7kRf
hales my ~ me ha nor a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales my ~ me ha new a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
halos my ~ me ha not a day a nary 7kRf
hales my ~ me ha not a day a nary 7kRf
hales as ~ me ha not a day a nary 7kRf

Iteration 200

hales as ~ me ha now a day a nary 7kRf
hales as ~ me ha nor a day a nary 7kRf
hales my ~ me ha now a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
hales as ~ me ha not a day a nary 7kRf
hales as ~ me ha now a day a nary 7kRf
halos my ~ me ha now a day a nary 7kRf
hales as ~ me ha new a day a nary 7kRf

hales as name ha now a day a nary 7kRf
hales as name ha now a day a nary 7kRf
halos as name ha not a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales my name ha new a day a nary 7kRf
hales my name ha now a day a nary 7kRf
hales my name ha new a day a nary 7kRf
hales as name ha not a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales my name ha now a day a nary 7kRf
hales as name ha not a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales my name ha new a day a nary 7kRf
hales my name ha new a day a nary 7kRf
hales my name ha not a day a nary 7kRf
hales as name ha not a day a nary 7kRf
hales my name ha not a day a nary 7kRf
hales my name ha nor a day a nary 7kRf
hales my name ha nor a day a nary 7kRf
halos as name ha now a day a nary 7kRf
hales as name ha not a day a nary 7kRf
hales as name ha new a day a nary 7kRf
hales my name ha now a day a nary 7kRf
hales my name ha now a day a nary 7kRf
hales my name ha not a day a nary 7kRf
hales as name ha new a day a nary 7kRf
hales as name ha nor a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales as name ha nor a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales as name ha not a day a nary 7kRf
hales as name ha now a day a nary 7kRf
halls by name ha now a day a nary 7kRf
hales by name ha new a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales as name ha now a day a nary 7kRf

Iteration 300

hales my name ha now a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales as name ha nor a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales my name ha now a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales as name ha new a day a nary 7kRf
hales as name ha now a day a nary 7kRf
hales my name ha now a day a nary 7kRf
hales by name ha now a day a nary 7kRf
hales as name ha nor a day a nary 7kRf
hales as name ha now a day a nary 7kRf

hales as name ha nor a day a nary sir
hales as name ha now a day a nary sir
hales as name ha nor a day a nary sir
hales as name ha not a day a nary sir
hales as name ha now a day a nary sir
hales as name ha nor a day a nary sir
hales as name ha now a day a nary sir
hales as name ha not a day a nary sir
hales as name ha now a day a nary sir
hales as name ha nor a day a nary sir
hales as name ha now a day a nary sir
hales as name ha now a day a nary sir
hales as name ha now a day a nary sir
hales as name ha nor a day a nary sir
hales as name ha now a day a nary sir
hales as name ha not a day a nary sir
hales as name ha now a day a nary sir
hales as name ha now a lay a nary serf
hales as name ha now a lay a nary serf
hales as name ha now a day a nary serf
halls as name ha new a day) nary worn
hales as name ha now a day a nary worn
halls as name ha new a day a nary worn
hales as name ha now a day a nary worn
halls as name ha now a day a nary worn
hales as name ha now a day a nary worn
halls as name ha now a day a nary worn
hales as name ha now a day a nary worn
halls as name ha now a day a nary worn
hales as name ha now a day a nary worn
Iteration 400

hales as name ha now a day a nary worn
halls as name ha now a day a nary worn
halls as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
halls as name ha now a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary word
hales as name ha not a day a nary worn
halls as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha not a day a nary worn
hales as name ha now a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary worn

hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha nor a day a nary word
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha not a day a nary worn
hales ex name ha not a day a nary worn
hales as name ha not a day a nary word
hales as name ha nor a day a nary worn
hales as name ha now a day a nary worn
hales ex name ha not a day a nary worn
hales as name ha not a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha not a day a nary word
hales as name ha now a day a nary worn
hales as name ha now a day a nary word
hales as name ha nor a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary word
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary word
hales as name ha now a day a nary worn
halls as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha not a day a nary worn

Iteration 450

hales as name ha now a day a nary worn
hales by name ha now a day a nary worn
hales as name ha now a day a nary worn
hales my name ha now a day a nary worn
hales as name ha not a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha not a day a nary word
hales as name ha now a day a nary worn
halls as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha not a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary word
hales as name ha now a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary worn

hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary word
hales as name ha now a day a nary word
halls as name ha now a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary word
hales as name ha now a day a nary worn
hales as name ha not a day a nary worn
hales as name ha now a day a nary word
halls as name ha now a day a nary worn
hales as name ha now a day a nary word
hales as name ha not a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary word
halls as name ha now a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary worn
hales as name ha nor a day a nary word
Iteration 550

hales as name ha now a day a nary word
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales am name ha nor a day a nary worn
hales as name ha not a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary word
hales as name ha not a day a nary worn
hales as name ha now a day a nary word
hales am name ha now a day a nary worn
hales am name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary word
hales as name ha now a day a nary worn
hales as name ha not a day a nary worn
hales am name ha now a day a nary worn
hales as name ha now a day a nary worn
hales as name ha now a day a nary word
hales as name ha now a day a nary word
hales as name ha now a day a nary worn
halls as name ha now a day a nary word
halls as name ha now a day a nary word

halls as name ha new a day a nary word
halls as name ha now a day a nary worn
hales as name ha now a day a nary worn
halls as name ha now a day a nary worn
hales as name ha nor a day a nary worn
hales as name ha now a day a nary worn
halls as name ha now a day a nary worn
hales as name ha not a day a nary word
exiting breed
gemini{46}%