Optimizing Genetic Algorithms for Use in Cyphers

Keith Diggs
TJHSST
Computer Systems Lab Research Project

2004–2005

Abstract

Over the past several years, genetic algorithms have come into wide use because of their ability to find good solutions to computing problems very quickly. They imitate nature by crossing over strings of information represented as chromosomes, with preference given to the more fit solutions produced. They hold great promise in the field of cryptology, where they may be used to quickly find good partial solutions, thus eliminating much of the intense manual labor that goes into identifying initial coding schemes.

Introduction

This document is the written history of the author's work on genetic algorithms over the course of the past year. The hope is to reach a better understanding of genetic algorithms through the results of intensive work on the subject in several different applications.

The Problem

Genetic algorithms have become very useful tools in the field of computer science, however, they are rather open-ended and can be applied in wildly different ways.[1] The problem here is to optimize a genetic algorithm for a cryptological problem.

Statement of Objective

The ultimate objective is to use a genetic algorithm to efficiently solve different problems such as the Beale cypher and the evolution of pi. Regardless of whether that objective is reached or not, the project should foster a better understanding of how to apply genetic algorithms to cryptology problems. The purpose of this report is to document the development and effectiveness of the project.

Scope

The project deals mainly with the Beale Cyphers as a tool with which to determine which properties of a genetic algorithm may be modified so that the algorithm may operate more smoothly. The project also includes an initial foray into evolving the value of pi.

Background

Historical Summary

> The Beale Cypher is one of the most famous unsolved puzzles in cryptography. Basically, 100 years ago, Beale buried treasure in a lake in Bedford County, near Roanoke. Three letters were written, encoded and given to a friend. Beale subsequently travelled west and never returned.
>
> Later, one of the 3 letters was deciphered. Beale had used a simple encoding algorithm. His letters consisted of a large list of numbers. These numbers corresponded to the first 480 words in the American Declaration of Independence. For example, if the document starts "The quick brown fox..." then 3 would represent the letter 'b' and so forth.

[1] Duray

> So, one of the letters was succesfully deciphered, but to this day the other two remain unbreakable. While there is a fair amount of evidence that the Beale treasure is a hoax, there is no reason to believe that the cyphers themselves are. It is also probable that the other cyphers either use the same document, just different encoding methods, or use similar documents and the same encoding.[2]

Preceding Work

I am not sure how much work has been done trying to decode the two as--yet undeciphered letters. Throughout the 1970s, one Dr. Carl Hammer used computers to analyze those two letters and found cyclic patterns in the numbers that suggested an actual coding mechanism as opposed to randomly chosen numbers.[3]

There is some evidence that the cypher is nothing more than a hoax. Kenneth Dobyns concludes after a lengthy statistical analysis that "there is no content to [the] cyphers" and that the numbers in the two undeciphered texts were chosen "substantially at random, limited only by the restriction that no numerical value would be repeated immediately adjacent or semi-adjacent to a similar value."

Theory

It is one of the great ironies of computer science that a machine built to outperform a living organism works best at times when it imitates the living organism itself. Charles Darwin's 1859 book *On the Origin of Species* laid the foundation not only for anthropology but for a great number of largely economic theories such as "Social Darwinism." The idea that the members most fit to survive would be the most likely to reproduce resounded in the scientific community.

It should come as no surprise, then, that computers might be able to produce end results as well as nature itself if they would follow this model.

A genetic algorithm is a method of computing that attempts to replicate nature's mechanism for evolution by organizing computer data into formats that are akin to DNA chromosomes. The idea is to have multiple manifestations of these computer data so that they might "cross over" the way that real--life chromosomes do. The "crossing over" does not happen randomly; the programmer has to code the program to do this, but he can choose to do it in a manner that takes two promising sets of data and combines them into what would hopefully be an even better set of data.

We need these "promising sets of data" because the data are supposed to be the solution to some sort of problem. Genetic algorithms are most effective in quickly finding good solutions to problems in which there may be several acceptable solutions.

---

[2] Matthews
[3] Krystek

The problem of decoding the Beale Cypher, then, would seem to be an excellent candidate for a genetic algorithm as described above. The search space is essentially infinite, yet it would be possible to program the computer to recognize good solutions as those with a high incidence of actual English words coming out of the translation. While there is one "correct" solution, but if I can get my program to come up with a solution that a human viewer could identify as reasonably close to this correct solution, then it will have been a success.

Design Criteria

Primary Criteria

- The project will be able to take the code of the second letter in the Beale Cypher and independently come up with a string in which an observing human can recognize the English message of the cypher.
- The project will use a genetic algorithm to do so, crossing over possible solutions to produce hopefully better ones.

Secondary Criteria

- The project may independently produce the exact answer to the second letter in the Beale Cypher, character for character.
- The genetic algorithm may be both time-efficient and space-efficient, allowing for easy application to other similar cyphers (even other letters)
- The project may produce a viable solution to the first and third letters of the Beale Cypher from which a human would be able to do meaningful and independent work.
- A genetic algorithm similar to the one used to satisfy the Primary Criteria may be adapted to other areas in cryptology and (this is a long shot) into other research areas such as artificial life.

Procedure

The project largely dictated its own future – since there was no initial goal at the outset of the year, the project was largely defined as whatever might be successful. At the time the author applied for the Computer Systems Research Lab, his idea was to create an intelligent computer agent to play the Japanese game *Go.* By September, the idea of exploring genetic algorithms had become fairly entrenched in the author's mind, but a search for applications was still necessary.

After spending two months tinkering with a program that evolved the value of π, it was decided to pursue the Beale Cypher in depth. The program

`decode01.c` was written to the point where it would begin crossing multiple chromosomes over each other and producing results that clearly followed progressive patterns. Substantive debugging was necessary to determine that this first version of the program was indeed producing progressive results as intended, but once this was done, it became clear that the program was in fact on the correct path. Programming would continue with the desired goal of successfully translating the numbers of the Beale cypher into a recognizable English text.

Once the functionality for crossing over was written into the program, work on the heuristic could begin. It is based on the relative frequencies of each alphabetical character in the English language[4] and the most commonly occurring words in the language.[5] The heuristic analyzes the translations produced by a chromosome and calculates how far the relative character frequencies of the translations deviate from that of general English, with priority values being assigned to the translations that deviate the least in terms of character frequency and also have the most identifiable English words. For example, a translation produced by the program that contained solely the letter 'z' would receive a very low score; one that contained a diverse mix of relatively uncommon letters would receive a slightly better but still relatively poor score; while one that had a well-balanced array of vowels and consonants with a reasonable amount of common letters such as 'e', 'r', and 't' and trace amounts of "phantom" characters such as 'q' and 'x' would receive a very good score. The intention behind the creation of this facet of the heuristic would be that it might be used in combination with another heuristic algorithm, which was later written as a word-searching function.

This word-search function was the most difficult part of the program to write, not because the goal was uncertain, but because it introduced an entirely new problem to the project: that of organizing an extensive list of data into the program's infrastructure in such a manner that it could then be successfully used to analyze translations of the program. The third version of the program code contains several different arrays that were alternately written into and out of the program using commenting. It took several approaches before a fully functional structure was built that could support an effective word-searching algorithm.

Most of the failed attempts at writing the infrastructure for the word search involved arrays. The first array created was a static one that merely contained the 500 most common words of the language in descending order of frequency, all written as strings declared between quotation marks and separated within the array by commas. I was later informed by my two greatest sources, Mr. Tim Wismer and Mr. Kyle Moffett, that this was considered bad programming technique.

The second attempt at writing the word-search infrastructure was a more organized (but also much more chaotic) dynamic array that grouped words based upon the first two characters therein. For example, the word "mile"

[4]Singh, accessed at Southampton

[5]World-English

would be found at index ['m']['i']. However, this quickly turned messy and the program rejected my multiple attempts at getting this into some format that C could understand. I had at one point a four-dimensional array, which C quickly rejected with multiple errors on every single line of the section of the code where this was placed. I spent many more weeks typing error messages into Google and trying to make some sense out of the forum discussions I found centered around these messages; I usually found it easier just to request the assistance of Mr. Wismer and/or Mr. Moffett, who were both very helpful. However, I eventually abandoned this approach as well.

I finally settled on a linked-list structure after a long period of flailing with this very error-prone multi-dimensional array. I pared the program back down to have just the initial list of words, from which the program would create a linked list as an initialization function every time the program would run. The program creates an array of linked lists that are indexed initially by the first two characters (as was the intent with the old dynamic array structure) and then by word. This way, the words "thousand," "three," and "the" would all be found in a linked list together at index ['t']['h'] in the array. The program was also rewritten later to alphabetize these lists so that the program might not count the same word twice if it contained a smaller word at the beginning (for example, "there" would not be counted twice for "there" and "the", although it seems that this has an insignificant impact on the final results. This component of the broader heuristic algorithm would count one point for every time the word-search function detected an English word within a translation of the cypher.

In the final version of the program, which I have included in this paper as Appendix 1 (earlier versions can be found at `http://www.tjhsst.edu/~kdiggs/code`), the `evaluate()` function can actually be written according to the programmer's desire with respect to how these two components of the heuristic evaluation (the character-frequency analysis and the word-search function) are combined to produce one comprehensive result. Other parts of the program can also be re-written to allow for more broad and diverse gene pools, more generations for the program to cross the genes over with each other, etc. This allows for empirical research on how different input parameters can enhance the effectiveness of the program in creating accurate (or not) translations of the Beale cypher.

Analysis

Functionality

Appendix 2 shows some sample results. The first part demonstrates the character--frequency analysis function – it shows five false translations of the Beale cypher and gives their variances. (Defined in the program as the sum of the percentage points by which the frequency of each letter relative to the complete translation varied from the ideal percentage defined in *Singh.* A lower number is better.)

The second part shows the word-search function, only recently completed.

The results shown include the debugging code I planted in the program. It shows the program going through the sample translation two characters at a time. Every exclamation point means that the program has identified the most recently displayed two-character combination as possibly beginning a word. Actual words showing up after exclamation points indicate that that word has been identified in the text and that the program has added a point to the translation's heuristic score.

It is important to note here that the above two components' sample results were not programmed to run in the final version of the code; rather they were produced from debugging code put into the third version of the program. This version can be found online at `www.tjhsst.edu/~kdiggs/techlab/code/-decode03-c.html`.

Results

In the last section of the appendix, I have included sample results produced by running the final version of the program four different times with different parameters, the specifics of which are included as headers before the results of each run. The strings produced at the end are the best or most accurate versions of the Beale text that the program thought it produced after the specified number of generations. The evaluation was conducted in each run based on the simple algorithm shown at the top. The two variations that I decided to include in the sample runs in this report take the word-search score and divide it by the calculated deviation from the relative character frequencies of standard English. There are two obvious conclusions that can be drawn from the results:

- It is important to give extra weight to the word-search component of the heuristic.
- Diversity of the chromosome pool is more important than the number of generations.

Note that the worst string produced is the first one, in which there are only 100 chromosomes to start with and the value of the word-search component is easily overriden by the variation within the chromosomes on the calculated deviation from English on relative character frequencies. The best string produced at the end contains long sections where the same letter occurs over and over again, and looks more like the result of a baby holding its finger down on a keyboard than something that might eventually become a beautiful English text. This is not to say that the other strings were beautiful English, but the reader can see that there is more balance between the alphabetical characters within the translation and that there are a few hints of actual English words being randomly formed but then preserved by the genetic algorithm, as it is supposed to do.

Conclusion

I have learned a lot with this project this year. I may not have even come close to translating the Beale cypher (I honestly didn't expect that I would, but I felt it was good to have such a lofty goal in mind throughout the course of the year), but I did learn a lot about programming, debugging, and even some cryptology methods. I do have faith that with further work on this project and possibly more powerful computers (definitely with more powerful programmers than myself), this project would be worthwhile to explore in the future. Genetic algorithms do seem to be a wonderful development in the evolution of computer science and all the fields therein, and I have come to see that they can move towards better results, even if they do not always achieve perfection.

Acknowledgements

I would like to extend a hearty thanks to a few of my classmates and my teacher for making this project possible. First, I thank Mr. **Chris Goss** and Mr. **Casey Barrett** for keeping a positive attitude towards the often difficult work that was required for this project, even if we did all end up getting distracted with the Helicopter Game a few times. Second, I thank Mr. **Kyle Moffett** and Mr. **Tim Wismer** for generously sharing their vast knowledge of computing to bail me out of several holes that I got myself into programming this year. I don't pretend to know C nearly as well as they do, and I am fully aware that I made a lot of dumb programming mistakes, although I was glad to be able to laugh about them with these two even if I didn't know why we were laughing and they did. Last but certainly not least, I would like to thank Mr. **Randy Latimer**, my instructor and mentor, for guiding me on both choosing and developing this project throughout the year and also for not going insane despite my addiction to the Helicopter Game. It has truly been a pleasure working in this class next year and I do plan on coming back to visit as I study at Emory University the next four years. Thank you.

References

Dobyns, Kenneth. "Beale Codes – Were they a Hoax?" *Genealogy, History, and Miscellaneous Material*. 1984. 25 Jan. 2005 `www.myoutbox.net/bealhome.htm`

Duray, Naranker. "Genetic Algorithms." *Imperial College of London: Department of Computing*. 1996. 25 Jan. 2005 `http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/tcw2/report.html`

Gantovnik, V.B., Z. Gurdal, and L.T. Watson. "Genetic Algorithm with Memory for Optimal Design of Laminated Sandwich Composite Site Panels." Technical Report, 2002. *Computer Science @ Virginia Tech*. 21 May 2002. Virginia Polytechnic Institute and State University. 8 Feb. 2005 `http://eprints.cs.vt.edu:8000/archive/00000555/01/gaCS02a.ps`

Huss, Eric. "The C Library Reference Guide." *Webmonkeys: A Special Interest*

Group at the University of Illinois. 1997. 14 Apr. 2005 www.acm.uiuc.edu/-webmonkeys/book/c_guide/
Krystek, Lee. "The Beale Cryptograms." *The Museum of Unnatural History*. 2000. 13 Jan. 2005 www.unmuseum.org/beal.htm
Latimer, Randy.
Matthews, James. "The Beale Cypher." *Generation5*. 2003. 13 Jan. 2005 www.generation5.org/content/2003/beale.asp
Moffett, Kyle.
Singh, Simon. *The Code Book: The Secret History of Codes and Code-breaking*. Unknown: Fourth Estate, 1999.
University of Southampton, ed. *Teachers' Notes to Accompany the Lesson Packs*. National Cipher Challenge. Southampton, UK: University of Southampton, 2003. *National Cipher Challenge*. 1 Oct. 2003. University of Southampton. 25 Jan. 2005 http://www.cipher.maths.soton.ac.uk
Wismer, Tim.
"The 500 Most Commonly Used Words in English." *World English: Test, Learn, and Study the English Language Online*. 22 Aug. 2001. 25 Jan. 2005. www.world-english.org/english500.htm

Appendix 1 : The Code

Included below is a copy of the program's source code.

```
/* DECODING THE BEALE CYPHER
Keith Diggs

Version 0.4
Version 0.3 - Searches string translations for English words.
Version 0.2 - Produces a bunch of randomized strings
  and has a function to cross them over.
  Also translates and evaluates the fitness
  of strings based on character frequency.
Version 0.1 - Produces a randomized string.
*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>
#include<math.h>
#include<sys/types.h>

/* THE MEANING OF THIS PROGRAM'S LIFE FOLLOWS
```

```
(these are just notes for myself put here by Kyle Moffett)
int * x = malloc(sizeof(int)*10);
x = realloc(x,sizeof(int)*20);
*/

// --------------DATA DECLARATIONS---------------
int* numbers;
int numberLength, stringLength, chromoCount;
char** letters;

struct hash_elem {
const char *word;
struct hash_elem *next;
};

//  -----------------STATIC DATA-------------------
struct hash_elem *hash[256][256] = { { NULL } };

#define GENERATIONS 1000

const char *keywords[] =
/*
List of most common English words accessed 2 February
  2005 at http://www.world-english.org/english500.htm
*/
{
"the", "of", "to", "and", "in",
"is", "it", "you", "that", "he", "was",
"for", "on", "are", "with", "as",
"his", "they", "be", "at", "one", "have",
"this", "from", "or", "had", "by", "hot",
"but", "some", "what", "there", "we", "can",
"out", "other", "were", "all", "your",
"when", "up", "use", "word", "how", "said",
"an", "each", "she", "which", "do", "their",
"time", "if", "will", "way", "about", "many",
"then", "them", "would", "write", "like",
"so", "these", "her", "long", "make",
"thing", "see", "him", "two", "has", "look",
"more", "day", "could", "go", "come", "did",
"my", "sound", "no",
"most", "number", "who", "over", "know", "water",
"than", "call", "first", "people", "may", "down",
"side", "been", "now", "find", "any", "new", "work",
"part", "take", "get", "place", "made", "live", "where",
"after", "back", "little", "only", "round", "man",
```

"year", "came", "show", "every", "good", "me", "give",
"our", "under", "name", "very", "through", "just",
"form", "much", "great", "think", "say", "help", "low",
"line", "before", "turn", "cause", "same", "mean",
"differ", "move", "right", "boy", "old", "too", "does",
"tell", "sentence", "set", "three", "want", "air",
"well", "also", "play", "small", "end", "put", "home",
"read", "hand", "port", "large", "spell", "add", "even",
"land", "here", "must", "big", "high", "such", "follow",
"act", "why", "ask", "men", "change", "went", "light",
"kind", "off", "need", "house", "picture", "try", "us",
"again", "animal", "point", "mother", "world", "near",
"build", "self", "earth", "father", "head", "stand",
"own", "page", "should", "country", "found", "answer",
"school", "grow", "study", "still", "learn", "plant",
"cover", "food", "sun", "four", "thought", "let",
"keep", "eye", "never", "last", "door", "between",
"city", "tree", "cross", "since", "hard", "start",
"might", "story", "saw", "far", "sea", "draw",
"left", "late", "run", "dont", "while", "press",
"close", "night", "real", "life", "few", "stop",
"open", "seem", "together", "next", "white",
"children", "begin", "got", "walk", "example", "ease",
"paper", "often", "always", "music", "those", "both",
"mark", "book", "letter", "until", "mile", "river",
"car", "feet", "care", "second", "group", "carry",
"took", "rain", "eat", "room", "friend", "began",
"idea", "fish", "mountain", "north", "once", "base",
"hear", "horse", "cut", "sure", "watch", "color",
"face", "wood", "main", "enough", "plain", "girl",
"usual", "young", "ready", "above", "ever", "red",
"list", "though", "feel", "talk", "bird", "soon",
"body", "dog", "family", "direct", "pose", "leave",
"song", "measure", "state", "product", "black", "short",
"numeral", "class", "wind", "question", "happen",
"complete", "ship", "area", "half", "rock", "order",
"fire", "south", "problem", "piece", "told", "knew",
"pass", "farm", "top", "whole", "king", "size",
"heard", "best", "hour", "better", "true", "during",
"hundred", "am", "remember", "step", "early", "hold",
"west", "ground", "interest", "reach", "fast", "five",
"sing", "listen", "six", "table", "travel", "less",
"morning", "ten", "simple", "several", "vowel",
"toward", "war", "lay", "against", "pattern", "slow",
"center", "love", "person", "money", "serve", "appear",
"road", "map", "science", "rule", "govern", "pull",

```
"cold", "notice", "voice", "fall", "power", "town",
"fine", "certain", "fly", "unit", "lead", "cry", "dark",
"machine", "note", "wait", "plan", "figure", "star",
"box", "noun", "field", "rest", "correct", "able",
"pound", "done", "beauty", "drive", "stood", "contain",
"front", "teach", "week", "final", "gave", "green",
"oh", "quick", "develop", "sleep", "warm", "free",
"minute", "strong", "special", "mind", "behind",
"clear", "tail", "produce", "fact", "street", "inch",
"lot", "nothing", "course", "stay", "wheel", "full",
"force", "blue", "object", "decide", "surface", "deep",
"moon", "island", "foot", "yet", "busy", "test",
"record", "boat", "common", "gold", "possible", "plane",
"age", "dry", "wonder", "laugh", "thousand", "ago",
"ran", "check", "game", "shape", "yes", "hot", "miss",
"brought", "heat", "snow", "bed", "bring", "sit",
"perhaps", "fill", "east", "weight", "language", "among"
};

#define WORDCOUNT (sizeof(keywords)/sizeof(keywords[0]))
struct hash_elem elems[WORDCOUNT];

/*int match(const char *myword) {
struct hash_elem *elem;
for (elem = hash[myword[0]][myword[1]]; elem; elem = elem->next) {
if (!strncmp(elem->word,myword,strlen(elem->word))) {
// The words are the same
}
}
}*/

/*string = "abc123wonder456";

const char *x;
for (x = string; x; x++) {
if (match(string)) {
printf("A match at position %d\n",x-string);
}
}*/

const double frequencies[26] =
/* This frequency table is taken from Simon Singh's "The Code Book"
  as referenced at <www.cipher.maths.soton.ac.uk/Teacherspack.pdf>.
*/
{
0.082, 0.015, 0.028, 0.043, 0.127, 0.022, 0.020, 0.061, 0.070,
```

```
0.002, 0.008, 0.040, 0.024, 0.067, 0.075, 0.019, 0.001, 0.060,
0.063, 0.091, 0.028, 0.010, 0.024, 0.002, 0.020, 0.001
};

// ------------FUNCTION DECLARATIONS-------------
void init_hash(void);
int* readCypher(char* filename, int *len);
int maximumValue(int* cypher, int len);
char* declareChromosome(int len);
char* translate(const char* chromosome, const int* cypher, int numLen);
double evaluate(const char* chromosome, const int* cypher, int numLen);
double variance(const char* translation);
int wordSearch(const char* translation);
char* mate(const char* chromo1, const char* chromo2, int strLen);

// -------------FUNCTION DEFINITIONS-------------

void init_hash(void)
// Sets hash[][] to include list of words
{
int i;
struct hash_elem *cur, *prev;
char a, b;
for (i = 0; i < WORDCOUNT; i++)
{
a = keywords[i][0];
b = keywords[i][1];
elems[i].word = keywords[i];
if (!hash[a][b]) {
elems[i].next = NULL;
hash[a][b] = &elems[i];
} else {
for (cur = hash[a][b], prev = NULL; cur; prev = cur, cur = cur->next) {
int cmp = strcmp( elems[i].word , cur->word );
if (cmp == 0) break;
if (cmp > 0) {
if (prev) {
elems[i].next = prev->next;
prev->next = &elems[i];
} else {
elems[i].next = hash[a][b];
hash[a][b] = &elems[i];
}
break;
}
if (!cur->next) {
```

```
cur->next = &elems[i];
elems[i].next = NULL;
}
}
}
}
}

int* readCypher(char* filename, int *len)
/* Declares and returns dynamic array representing numbers read
from file; changes value of len to represent the length of
this array.
*/
{
int* returns = malloc(0);
int index;
FILE *infile = fopen(filename, "r");
for (index = 0; !feof(infile); index++)
/* This for-loop will actually append a zero at the
end of the array at the end-of-file marker, I chose
to leave it alone since it won't really affect the final
outcome of the program.
*/
{
returns = realloc(returns, sizeof(int)*(index + 1));
fscanf(infile, "%d", &returns[index]);
}
*len = index;  // !!! -- FLAG -- !!!
//  (this can be a trouble spot)

return returns;
}
int maximumValue(int* cypher, int len)
/* Finds and returns the maximum value found in the cypher,
used by the program to determine length of each chromosome
*/
{
int i, max = cypher[0];
for (i = 1; i < len; i++)
// Assumes cypher has length of two or greater
if (cypher[i] > max)
max = cypher[i];
return max+2;
/* This must be augmented by two, or else the program messes
  up.  It prevents random numbers from being appended, which
would result in a lot of wasted memory and time.
```

```
*/
}
char* declareChromosome(int len)
/* Declares and returns a string of length len that represents
one chromosome; character at index i in this string will
correspond to translation for integer at numbers[i].
*/
{
char* chromosome = malloc(sizeof(char)*len);
int i;
for (i = 0; i < len-1; i++)
chromosome[i] = 'a' + (rand() % 26);
chromosome[len-1] = '\0';
return chromosome;
}
char* translate(const char* chromosome, const int* cypher, int numLen)
/* Translates a cypher using given string as the key: a number
  n in the cypher will correspond to the character at
chromosome[n].
*/
{
char* trans = malloc(sizeof(char)*numLen+1);
int i;
for (i = 0; i < numLen; i++)
trans[i] = chromosome[cypher[i]];
trans[numLen] = '\0';
return trans;
}
double evaluate(const char* chromosome, const int* cypher, int numLen)
/* Returns the numerical evaluation of the translation of a cypher
  as produced by the given chromosome.  Algorithm not yet defined.
*/
{
char* trans = translate(chromosome, cypher, numLen);
double v = variance(trans);
int w = wordSearch(trans);
return w*w / v;
/* The above line can be changed based on how the
 * programmers wants the program to evaluate translations
 * produced by the chromosome pool.
 */
}
double variance(const char* translation)
/* Calculates and returns the raw variance of a translation's
  relative character frequencies from that of the English language.
*/
```

```
{
int frequencyTable[26];
int len = strlen(translation);
double var = 0.0;
int i, ltr;
for (i = 0; i < 26; i++)
frequencyTable[i] = 0;
for (i = 0; i < len-1; i++)
{
ltr = translation[i] - 'a';
frequencyTable[ltr]++;
}
for (i = 0; i < 26; i++)
var += fabs( (double)frequencyTable[i] / len - frequencies[i] );
return var;
}

int wordSearch(const char* translation)
/* Identifies the number of times the program detects actual English
  words (provided in the source table) within the text of the
translation and returns the number of occurrences.
*/
{
int transLen = strlen(translation);
int points = 0;
int i, j, tokenLen;
char a, b;
char *sample = malloc(sizeof(char));
struct hash_elem *curr;
//printf("%d\n", transLen);
//printf("Text to be searched:\n%s\n\n", translation);
for (i = 0; i < transLen - 2; i++)
{
a = translation[i];
b = translation[i+1];
//printf("%c%c ", a, b);
curr = hash[a][b];
while (curr)
{
//printf("! ");
tokenLen = strlen(curr -> word);
sample = realloc(sample, sizeof(char)*tokenLen+1);
for (j = 0; j < tokenLen; j++)
sample[j] = translation[i+j];
sample[tokenLen] = '\0';
if (strncmp(sample, curr -> word, tokenLen) == 0)
```

```
{
points++;
//printf("%s ", curr -> word);
}
curr = curr -> next;
}
}
//printf("\n");
free(sample);
free(curr);
return points;
}

char* mate(const char* chromo1, const char* chromo2, int strLen)
/* Uses crossover algorithm to mate two chromosomes to create a new
  one.  Crossover site is determined at random.
*/
{
char* issue = malloc(sizeof(char)*strLen);
int cxSite = rand() % strLen, i;
for (i = 0; i < cxSite; i++)
issue[i] = chromo1[i];
for (i = cxSite; i < strLen; i++)
issue[i] = chromo2[i];
return issue;
}

int main()
{
int i, barracuda, dad, mom;
double maxEval = 0.0, minEval = 100.0, evalTemp;
/* maxEval is the highest evaluation number encountered in the array thus far.
 * minEval and maxEval are used to set a range to determine the probability that
 * a chromosome will live to the next generation (as opposed to being discarded).
 */
char* translation;
char* offspring;
srand(time(0));

// ----- INITIALIZATION -----
init_hash();
numbers = readCypher("code2.txt", &numberLength);
stringLength = maximumValue(numbers, numberLength);
chromoCount = 1000;
letters = malloc(sizeof(char)*chromoCount*stringLength);
for (i = 0; i < chromoCount; i++)
```

```
{
letters[i] = declareChromosome(stringLength);
evalTemp = evaluate(letters[i], numbers, numberLength);
if (evalTemp > maxEval)
maxEval = evalTemp;
if (evalTemp < minEval)
minEval = evalTemp;
//translation = translate(letters[i], numbers, numberLength);
//printf("%s\nHeuristic value = %f\n", translation, evaluate(letters[i], numbers, numberLeng
}
printf("Best value so far: %f\n", maxEval);
// ----- BREEDING -----
for (i = 0; i < GENERATIONS; i++)
{
do
{
dad = rand() % chromoCount;
evalTemp = evaluate(letters[dad], numbers, numberLength);
}
while ( (double)rand() / RAND_MAX > (evalTemp - minEval) / (maxEval - minEval) );
do
{
mom = rand() % chromoCount;
evalTemp = evaluate(letters[mom], numbers, numberLength);
}
while (mom == dad || (double)rand() / RAND_MAX > (evalTemp - minEval) / (maxEval - minEval)
offspring = mate(letters[mom], letters[dad], stringLength);
do
{
barracuda = rand() % chromoCount;
evalTemp = evaluate(letters[barracuda], numbers, numberLength);
}
while ( (double)rand() / RAND_MAX < (evalTemp - minEval) / (maxEval - minEval) );
if ( evaluate(offspring, numbers, numberLength) > maxEval)
{
maxEval = evaluate(offspring, numbers, numberLength);
printf("New best!\n%s\n", offspring);
}
letters[barracuda] = offspring;
free(offspring);
}

free(translation);
free(numbers);
free(letters);
return 0;
```

```
}
```

Appendix 2 : Sample Results

Character-Frequency Analysis

```
String 0:
upombjqbubbpcdmqrbcwnrimpptgvxqsmggoqxrmavbeyquptznhqwvgnomlledyljo-
dihpicspakjseihnzdihtwtejzhjkszndhenjzcmjmgnwnaxtkemixqszzdjrdzumfk-
xemjqardxfguaqbmzactzbwjubkrnpbnwnymoqpnxxjxoklplpiksgtmkdfzvgmrwld-
mbuzasixitkxezhqrpkfzplzmrldpyibzdcbxnyfiieojneacqgegrfuksaaridsniu-
kvurdfvrokvpnovtuoyqaqqcjbgcoowblssoxqhmaebpsyiooifobvrkwzmnnkoadgo-
awvoxzpoupxidhnuilntjmhwbsmbxtpzrmoqnffhvesalfwusjpcvwayqnzpiqpzefs-
ukzdhdvkqbgmtsbxpayosnpivgzwmeqgopkyufiliqxekbdabbqtqfdnncmcgcixuux-
ochzkxzqicukdxcyohedxgpzpsjomelcgsesrfjbhfnmcpmszqwwzlxqfgetkrwslal-
efwimbvafnmymdukcfhsnnygarcueehildqzgozmbmknpgxtngnavlhvelsiszqddjd-
lxexaqjphroafuphpcqngefowehzjsevrlurbgglxuneodmffeuoizceglgpdmmuxhl-
zptmmnztdchlklfyufcyloecrtpmqyzrntdbidwohdzrperjlujyiqbbjspzsptfkwi-
sbfglihezowizquaakdbtvrungnusmeknjtauaftobeqtyqtktudqnzfvpbnbfaqrtr-
lvwgmzmeskwlwrhbhvboqqpetwgjnzcywyhkxtorgkccbkekhhaxxpdqojbbkecjejt-
eekxkwapymtktalqzcwrqhsuuywdfhylliiyfknfwirpkeijhgaznuvhvrmcalnovxm-
ckbjgjbytfgcooeqbyllvfxxfkmtgmhkwiuerxcmckqtyulcvwnqdmqjxcefoopmwjr-
q

tspgxqzbsdoibqrpxjednybtejmaiqkmlkqbuohsmkwmobboswsnjqmnsmirqvnxqjq-
gpmucpelgqoevszqsqdrjznocgsionlmhzdmflpcufmwqymqdsedfmqipthsttzudnj-
irubarntxmoaeyexdmnxcvarbbowsdnhqnendhfxargqmuyqwecepgnlnmjlfbozvnn-
qmszaneqgqjrumdomrtixdyemxqpsymqwemhyymcynmzencnnnkrdskfequmemarpte-
qttsgaoayfbqzyqraxugxqckbqocmdtngzweqqsddviybmgquanmgxnqrpinxzplqbi-
gbepypgdxyqdnijnounmbgyiabbmemyhynkdsbrroelbsuqoygmnmbqqnnwxqzpbrgg-
byqsqbqomkasnhymbxogegqsonbfxqvmbgutjneboasrdpndngglhuregqwifrejmgu-
pqyryoixmqirobbqbnpytbtunygdgnqynqdsvhnrnpcmygvbqqxnxmjrntiqmwndnrq-
bqmmnyznimnbpccgxdndgrkfnneqzdqcqomfmncmlrcunakndmjmgtsnrsmgzkfnzos-
mouaeefuyxqaigarrupipyqlsqgisngyrwxrisbnmerqrmjybnqezsgddgdsnmiqxum-
vjlfrqzczwqqxxbwcpiyqnrnrohrmsdmqzdrbjbpburedkmalghsdvdmmnvtnsnumkr-
dibreadenvqahyimsubyrptrau
Variance:  0.766093


String 1:
gtcletgeprgknvdritcfximmwbyvnplvlpgrimvafekvzpojiqqgacuzeuutmfoxwvq-
fjnfprrkthaerqvzsztrfolycsnbpkrwvedmywwtdyxxouwjuqazgoykinlzxcvsgaf-
gxdzcczzstwbpoepwupgecrebvbvdcakzgldkkdcgbgxrmopgdxmhoqklsgowiyxoma-
ywddeglezxsofylrhbismcabyibvznvxmbcsjnwjmpxrpllwmtqbxqcvzdsytpxfsax-
cpunclmtaxeymzpnyfpveunenfcuxesbvooxzbtbazzmbobzwsvcnkhcplzpprqkhhk-
```

ikdjkckzdbafxtczimimetlvkenxluhtgmetqrswsubrpdszrbmvuzshggerannjbtc-
rkuneqqvgtqfnrtkntduzjaslphuqbwinsxsjoprjfwywsjmlmgmxjgiyofrrbbeuyy-
foowatuysmjezxmnuvufwkknbnohhogofukfphfjtrptocilzcqxoampnazxrfnwbad-
qhkcabrvstedvitsxvfokhnjasxwtxbmgmoipfghbkkwudqtaxilfvwhqvelshzatnj-
lvpswbduvimplkzyrwuymqeakmzlhnwuklnioselaqcncbltygumwamiomtybrsnefv-
szzecrgpukbpihjwfmjnaxjyaatnhykzajfccorwysoibzeilpymmhmojfbsglsgwzi-
znbxlulvxlcgwrgkgpxuaeytkllsikajaeinyujxflfdelqkaphbtfugqhyatbltfuh-
fqscxfkajvquyhbbbivhaehdxiotpkcuauzxzjaigseeahfbpckqgrtgcjzrubmwvlu-
wwugcmljmuopktbcztxfxigochczepvclpjqdwzsqnjagleigdpdmwtqfwrmnmozdzr-
jvqbmdmomxvwdylikjcbpaudnhtoktfvovwcblrpimlomwwxiyazawcodveqqklfhkj-
i

urwcmaqtvocmerzjpdrzewgyatyatabdtsirowfktpcuckerkcvzatyekzpzvnimivo-
cwlpnjvtcpwdeeeikcsiaqmurgrtrqtyvsstlmrwustzvxlgskvbttnbjivkyusbsxa-
txogwziiotcwfdapldxmwnwiverfolxvvwapogsmwzccapzvzacrbgxtpvdlseqeexq-
ghkewxfapltvpdzuahuppzxvgpcjrwaacrzgzxynwulqriweqhspleftdryabdezbya-
cuirgewwxseawzaifmbcmynpgrrngoumcqcxclvofetxgtccpwxlpmivzjtqmqjwgep-
pevjxtvtenrsltbpcuxaepzmfeglatzvnisavgvzeflnonvczplpvggthlcmasjnvpc-
ezlvtgrrtbwvmenanpqpdccowinsmananppidqaecwrilkxfmcptvbvvdcfttvvadvp-
jcxzzrmxttbhcgeiexbzieyoxncsguozxylkegqzejpaxcnktcqwaldzuytcacxlqvi-
gazdmdwupdxewcncmfxopipspmalszbwrcystiwtmtrpqemulavtcqkxirlcwvtqefk-
twuedatpzmafmgeviojtwzrwrcctexpdpcmvmvgxtdzrvltzewvxwkpzlcskpzptaut-
edlstlwwqcvtegecnjtxcxiuzuvpdrzhiwaiktebkpialblelcvknetddxnyiolpgbz-
zpkiaffdmwcegxmakpgxzjizeg
Variance: 0.714607

String 2:
vcxfqktdhqcrqeqtasheplwgxjnftwpqznypzrsijwzcapvbhehxrffposwkolapayf-
aszidwifyzabhgigynlpeflotyqjzrqblplonvumvwnedykdnxsrcfhxdyierahcstq-
fqktljjpmhbrwajoeqxctvmamnhqfchmttggcpyqyzhuasigiikedxesmnkrsrfnllt-
pcriatpvvkfbsnmytlenxuxqmpxzcjutnobojtlgdsjxgxwbiaohunziexjgigcvwdl-
fwwlzpuzxsvycxomublegkoouwqqswbzsamuritocqofperlheqnrgelewdwshxnhjh-
arcouuebliuxpapfrxlfbhizapwnyhwyaaouwuxhcrgusxblumqxtbyvswkrdirgkia-
gcaqhryblwezsqpslsqjlntesexyofbvibnpulsihwhcoyuzrmkezeiuifuynxtvaim-
xuefdcmhqkbrdocjpitlsbiqofmjiuxfqbkwfydvkfoqvszkatyuuglinxtvtqaktmg-
yklvxqjnlcmyefwzzemjrlcpgupqneylqkgncsaqwnqcuodwspflciajcsbswcdmoma-
qgcicpyemmkigznsexuncmouiqzvgnxxtzhxrgefuondpcxtcrigezcmrdjyrixmiek-
bmqgjgvmvxjrbazheajtsneqgnnsxteygugpabdyamrcosjtttolisdofqgcjncrhig-
kljkmxbomtzhouvzcpesvuzzgocxxmghyrtmpuadqbkgwhiywmrthstpgxodlunloix-
ffzixbudxdlxbyqvhiqwqpmwajlmxtlcylmvoizllmimmzhtjzrcoeyrnlfmeqqfdca-
rmbexqnlcotwzupbjvbckmhwtzoydsaweebbuqnygiufemjphmrszaquafsdxubeych-
uswtyepglbpalbsfcsvwscqxclbgldngwkbwqqcbutehvwmzqjylooksblynpnvnzwj-
s

nixirresqapwdtjbwvhnolcndkfclrhkkfzqvhmxmzfwxzdpxfqgaaforrdjetlrsya-

ixfhqbckiphlwbtsxlnaaeoswyilpwkfiynmporxwnmbepftnxcoimorbhirnnymnea-
lnvckjlhfmxclqdwiqprutbandpeaieiebsoaxnrkjilxhaebshhiypkoevqndftwph-
tqxtbelrnpkshqnsxtndwnpcswlbilxrfhrxapfqlyfehluowlffipqhltuxhqajind-
lnhiyahcpndrearajrmirhqzctpqsanoiefqlpqaawlpcmilhkpznrlejblhrebatdd-
nqcbpcfovjtnxljoxwpidnawjdcfdmaijlftqcsjwlqrmmexanzoectalxfrrybrsni-
dapqactpmhkqohjxrwfnlilahlrnrrtxrnhhvhddxkiaireaoinkimscoleliscaqfh-
blpjapwumartxcdsqpjahdnvejinyyaaehirwxhjobxxpitzaldbuzvjynllxfpiwsz-
crrkoqeydqpdxhqiraeanlznoodpynixtxwnmlumozwhhaeyixymiereaiziebihtbr-
mhwaldiharrjdyasavblxataililbenqffrswqcemljtsfkadbeqernniinrorsuuam-
wvqnzpeuefeavfdfqblplelyjpifqinqsetazkqjzhadihzaqiirqwokqptnlmxhshj-
nszadjaloslaxpdxxhcpjbhjgv
Variance: 0.544788

String 3:
oovkhzrexfeqrcbfslxuzxhqmcdoaoiocdzkfsoeyuupwvupjtlkqsaewdswtakwgli-
newrcrmrpjnguirealeehjzffzqbhdjuhhmlaaesjtagbskbfrinchshikkqovkxfyl-
hypzijaomuaoztyowfhfntqfjnqjsouapjbxubnhxpxzjvnhcwmrqexbspkmfeouqqt-
mtitrysqhpfotccluhkxbzipgoebguwtpfogdhymygaskedenpdqqngwcmzkgwdxbsg-
gbgszmurwywcongedtkaxybigxneafkigqaidwbbxyellmprhcrhcurjuenumxfuphc-
seewbecorqfixjbgmwxvqeimqhtkwbprfvnizryshddgnenbcmwuqgjincvjfmcmhru-
iiuapydwnklooxmlqsuyfyvqehvqomnyvhbkhgiuqtkhtwslqolxmhqsqllhxyfshgf-
rnplfjymewgpnucmhjcbanmhlmofuvmlkcyplyervkikhkwqwzryoefbqvhmstxcvyu-
iwybtkldtwcjsdbqrfxvxucknxjpvjjfgjjbwuhpqjzkocchjbcgwgsjgcablmitxrv-
tmelepmrdotkxxpftvyebceepspjqgglumybbksgalszkhfddfjghonyicjykpkfdig-
evakvndxymebpjnysdlslpckztwgxgmbboozctwafcdxnqvhtizfxbryvnfsttvxijw-
mcumiyrflicsdmtklvbjsqqmmkhjuthgxdshblomtqgzezjpwnzodpapbkavfjdcmxk-
pjadetjdxlpnhcoyhfyzhkzeplisxfepqgsvzcaxppmwrcuzhvaqfzvxmfrkmxzddta-
dxacmqqljsfkzckqkmnjyuakixlncfnfcqjrgzcrrjetmolycahbujlegzsiehqixbz-
fbbxunbnbryzuyixurkzzjrhpazyybzdcdcyqfmrzmsvlcshtcgsozbdabdyeddiiig-
a

fmmdsqtlokkhefapojulwyeddzoxeqqpwfffuhxhshssvuekhsoenpowdncaaaxsalt-
dmkxrppwdvhmugnahipsntldrzmekzworzpsmtjvgxsgawkrphpfrsbppjrddfzjpgn-
euuekaxjhsveaudokbzseaesnekuqkgjauiikkxskaditxwagixuqzzwisjtxeinuzl-
rphnegaqdizoxbldtufcolwpfoipmytqsunkwworysktuxewzuflkhlumfdtqbraqdd-
ifjmzrhewxeqwwqsysjdssrhefkrfkfldtsyiioknuewesdixkzcdsxaapelstpgrec-
dhppwooxqufpeebivgzeedwhyeekdswjuxftoeoafatcafavwdciserpuessqapcodd-
ewiopefksqkolputcoidmdiqhxcxsqatcdxjjldevkmskqgnlddwrjopciueropnbox-
piwawkhzsppuveeahzcwjedugudpzstwgskduklawphtwdaupicuacjasdeitszkzof-
eqnpluwscbzemxrdsngkdxhxildialwvfvtxsxestrrxlrrsktlsdxdgsmcdwwrlntd-
shgrmdrxwsqyyzrosupemwfgmideggdulssohoegsmafokzweuaywddlkdpdiniiahs-
ujtxriwetsapqhesrpewigxsahjlbmlpawtsuzhcuxsdkqcrtdrdauxpbzadxaexfqa-
liusdynmlfirkwythxewapjaxo
Variance: 0.628895

String 4:
pstqhllspdziiicwhdbjgfjkknusxbsovlfcyswnyyxggzepefznmizwxuquvklqxst-
vkrkipjqviulpckcoubnsvdosqzjnrelevvonhfirbvgfhkwcmlujokcgknudtfhocy-
djdobgjjotukxivtrldvspiotbtcsxavhfjkovtcqpopxjirwnmqdxewaxasvbqeibq-
wyjaobofaznrwbgogftcfscanfqtotjmsvniyboyddqbgypmdiokbtkqacmqwvdqssa-
qtqpyuhzafpnkxduayhrzjdphauzvmcliscgmjgpqxecuhzxigqhrvybvvcqheerxgy-
lsgakdgnzomwyumhlhfofcrxkvdbsmbeehgomnbmdpkbjzixiplpscpeyshsejwksfz-
hucvxsfadgkapbmgvqvzoojhuuffomevgiqfcyhggtixvwesnzubpdkkxspnevjndbu-
gbcmivwfstllglfjckwmhobuuwfhcjcikgwsdsawlnjtyqdbazpjptfjrlttuvbgdxb-
gsdefqpzpfesffjqucwfujynfvrlyqohirnoidpxktrqyaiufezzpyouthiuywbioow-
wtnvegowgqeavlawaanxvuhpsfsbvjzrcnogvcecvjeqwepygcvcyettkmufvvzzkpi-
gsmkpvohtvwsbapdzuzuglqnilohyepgyeqnbguvdrnftckuyjoeweuesioqmfykjqx-
lwuganvhgxrdwdtczywgsfuktziejydwwxcwmxdtwxyscrxbrtjjadtuddaobfkyfow-
rlcmjzkcdcbetuqfxtatyetofyyfdmcvzehjfuniwoorkewjyxcydvniwlobaswbwgm-
bazjypxrceqlcnocslpahdqjfkhggptjruurmujoyzzcopegbuikzbteldmuufdmcyd-
osocsndubtyhuusevtazefmyztenxetlyhpqvtmyokikhaoctqezxsaxlemlkfyipoa-
k

cjkmsmfdolwjswjpbrpuxozuclqqbmtdunydeiwciniqtxscciocuiqxruijrxfsksi-
mkqpipgumzivylskcbxhufoupfjbcmuqkqxiuvsaqcicrqqlxcgvfidzpekrucqtxgu-
beezkjfeoitekvcbncgsfxmhbscjsngkrlljlncskjmbfpgrclbpofgujfrwcstsygz-
lxcsmgkmlslwpcuufgcibuqgobbpjofmipungqqiohqfpffxmjndnvytvwafkcojouc-
bcejfoieqcsmrgmhystmskinzwciolcomfijbsolvybqzimbpkgvlsfrjpbzsfpxlsi-
lhgpqssoftwxybjjtqgnslgjyszqcigktfnqozwjmkwrczrtglvjfzlijyismoprwlm-
sgsoizwcitkooxtfrbtlvmbsifrcsmxfrlperzcstkjhnigvomluktwgabjbfwgucsp-
pbqjgcjpiizgtzskhgngesuegtmxfhiggknrynzjxpdfqmxxibilvvrjhubbfignmwy-
zmudovrhicgskbimsvgllfncjocsoujawtbciffivzppzoqhnfsimerghjvmrafzsyr-
iiqovcfpgsmydfowhepbkgwxjbmblglvdiswjozgivjwwqlgslrjrrlunmxrjuhovdi-
yrwczsrffirifdsiipbqbgfhjckdcjuxkrqhxlhnxphcntvowmkrvyodcgxufcypotj-
uhxhcyvvovbonqdfcpzqjpejzp
Variance: 0.800033

Word Searching

Text to be searched:
spqlovrfbchlrrtfjabsqamvbvrqcvcpbsrrclubznqlnyrlbqbgkbrqqevtvjloyyy-
lquldflblclrr wnybuxkkrapuwpclsbrauxzgahalizuvnuqxblfuzwefhaqvsuoxd-
kcqcmltlhkzntuubjkbwoijpkrr lxnkdsvykecdioltlunluvuknbmwwbecayirbnr-
weqwbnpduvwjvclbspndsvjsnljjufpanvqbedunr dazurbliqshsiksetrrtnrbit-
mvbushpwiltnirvguvkgooloqdnmrldjcsalrqsujbckrcnmzlullwn wolvtfceorf-
tqrvwwlfnexoekrxjcjenlwerwulgrmubzusklsrbmctluywpqvnuwnecmqbhjqovyf-
wc wlrujbbmrlzclbauknwjbwrlunllwiovjnwwlhaebrnlpkksdkalwbaocliuxcuc-
lkbxlfuntulluzbe dnmrywwauhrvcdklxwzyudqkqrdetqfhnnljybujyjnatzvcun-
qwkscrmvepaugzvbwrqndlokdcweni eabjyslarnuizlizaruleilzknyzlvqdkpnl-

gmuenhqzllirbuluovgkwickcfcqurtpulcwdwuiqocl bmdzrtrcuvuryvsgqwsklx-
qeeabjvzrayirjgirqvbetrqdfcnudeztfsibpswygrkyvwaylkbkcniylaqbropbwj-
vlpjljctsaykbgkrakuidnknblmntfhtis

sp ! ! pq ql lo ! ! ! ! ! ov ! vr rf fb bc ch ! ! ! hl lr
rr rt tf fj ja ab ! ! ! bs sq qa am ! ! **am** mv vb bv vr rq qc
cv vc cp pb bs sr rr rc cl ! ! ! lu ub bz zn nq ql ln ny yr rl lb
bq qb bg gk kb br ! ! rq qq qe ev ! ! ! vt tv vj jl lo ! ! !
! ! oy yy yy yl lq qu ! ! ul ld df fl ! lb bl ! ! lc cl ! !
! lr rr rw wn ny yb bu ! ! ! ux xk kk kr ra ! ! ap ! pu ! !
uw wp pc cl ! ! ! ls sb br ! ! ra ! ! au ux xz zg ga ! ! ah
ha ! ! ! ! ! ! ! al ! ! ! li ! ! ! ! ! ! ! ! iz zu
uv vn nu ! ! uq qx xb bl ! ! lf fu ! uz zw we ! ! ! ! ! !
! **we** ef fh ha ! ! ! ! ! ! ! aq qv vs su ! ! ! ! uo ox xd
dk kc cq qc cm ml lt tl lh hk kz zn nt tu ! uu u b bj jk kb bw wo
! ! ! ! ! ! oi ij jp pk kr rr rl lx xn nk kd ds sv vy yk ke !
e c cd di ! ! ! io ol ! lt tl lu un ! ! ! nl lu uv vu uk kn
! ! nb bm mw ww wb be ! ! ! ! ! ! ! ! ! ! ! **be** ec ca !
! ! ! ! ! ! ay yi ir rb bn nr rw we ! ! ! ! ! ! ! **we** eq
qw wb bn np pd du ! uv vw wj jv vc cl ! ! ! lb bs sp ! ! pn nd
ds sv vj js sn ! nl lj jj ju ! uf fp pa ! ! ! ! ! an ! ! !
! **an** ! nv vq qb be ! **bed** ! ! ! ! ! ! ! ! ! ! **be** ed du
! un ! ! ! nr rd da ! ! az zu ur rb bl ! ! li ! ! ! ! !
! ! ! iq qs sh ! ! ! ! ! ! hs si ! ! ! ! ! ! ! ik ks
se ! ! ! ! ! ! ! **set** ! ! e t tr ! ! ! ! rr rt tn nr rb
bi ! ! it ! **it** tm mv vb bu ! ! ! us ! ! **us** ! sh ! ! ! !
! ! hp pw wi ! ! ! il lt tn ni ! ir rv vg gu uv vk kg go ! !
! ! ! **go** oo ol ! lo ! ! ! ! ! oq qd dn nm mr rl ld dj jc cs
sa ! ! ! ! al ! ! ! lr rq qs su ! ! ! ! uj jb bc ck kr rc
cn nm mz zl lu ul ll lw wn nw wo ! ! ! ! ! ! ol ! lv vt t f
fc ce ! ! eo or ! ! or rf ft tq qr rv vw ww wl lf fn ne ! ! !
! ! ex ! xo oe e k kr rx xj jc cj je en ! ! nl lw we ! ! ! !
! ! ! **we** er rw wu ul lg gr ! ! ! ! ! rm mu ! ! ! ub bz zu
us ! ! **us** ! sk kl ls sr rb bm mc ct tl lu uy yw wp pq qv v n nu
! ! uw wn ne ! ! ! ! ! ec cm mq qb bh hj jq qo ov ! vy yf fw
wc cw wl lr ru ! ! uj jb bb bm mr rl lz zc cl ! ! ! lb ba ! !
au uk kn ! ! nw wj jb bw wr ! rl lu un ! ! ! nl ll lw wi ! !
! io ov ! vj jn nw ww wl lh ha ! ! ! ! ! ! ! ae eb br ! !
rn nl lp pk kk ks sd dk ka al ! ! ! lw wb ba ! ! ao oc cl ! !
! li ! ! ! ! ! ! ! ! iu ux xc cu ! uc cl ! ! ! lk kb bx
xl lf fu ! un ! ! ! nt tu ! ul ll l u uz zb be ! **bed** ! ! !
! ! ! ! ! ! ! **be** ed dn nm mr ry yw ww wa ! ! ! ! ! ! !
! ! au uh hr rv vc cd dk kl lx xw wz zy yu ud dq qk kq qr rd de !
! ! et tq qf fh hn nn nl lj jy yb bu ! ! ! uj jy yj jn na ! at
! **at** tz zv vc cu ! un ! ! ! nq qw wk ks sc ! ! cr ! ! rm mv
ve ! ep pa ! ! ! ! ! au ug gz zv vb bw wr ! rq qn nd dl lo !

! ! ! ! ok kd dc cw we ! ! ! ! ! ! ! **we** en ! ! ni ! ie
ea ! ! ! ! ! ! ab ! ! ! bj jy ys sl ! ! la ! ! ! ! !
! ! ar ! ! rn nu ! ! ui iz zl li ! ! ! ! ! ! ! ! iz za
ar ! ! ru ! **rule** ! ul le ! ! ! ! ! ! ! ei il lz zk kn !
! ny yz zl lv vq qd dk kp pn nl lg gm mu ! ! ! ue en ! ! nh hq
qz zl ll li ! ! ! ! ! ! ! ! ir rb bu ! ! ! ul lu uo ov !
vg gk kw wi ! ! ! ic ck kc cf fc cq qu ! ! ur rt tp p u ! ! ul
lc cw wd dw wu ui iq qo oc cl ! ! ! lb bm md dz zr rt tr ! ! !
! rc cu ! uv vu ur ry yv vs sg gq qw ws sk kl lx xq qe ee ea ! !
! ! ! ! ab ! ! ! bj jv vz zr ra ! ! ay yi ir rj jg gi ! !
ir rq qv vb be ! ! ! ! ! ! ! ! ! ! ! **be** et tr ! ! ! !
rq qd df fc cn nu ! ! ud de ! ! ! ez zt tf fs si ! ! ! ! !
! ! ib bp ps sw wy yg gr ! ! ! ! ! rk ky yv vw wa ! ! ! !
! ! ! ! **way** ! ay yl lk kb bk kc cn ni ! iy yl la ! ! ! ! !
! ! aq qb br ! ! ro ! ! ! ! op ! pb bw wj jv vl lp pj j l
lj jc ct ts sa ! ! ! **say** ! ay yk kb bg gk kr ra ! ! ak ku ui
id ! dn nk kn ! ! nb bl ! ! lm mn nt tf fh ht ti !

Translation

word count / variance
10000 generations
100 chromosomes

rorsoscooaiomzauoodoobybaocmrsustastoosmouzoncmlozooobcoonsasroowgbo-
rsmoumtobonzrawomnaocmszoonlmtctmnoaossmooosssbnomxlossumtobnmsnlont-
oyoaommowrrcwoltsoorsaomlobllosboaamoooaomomnsooxdcostawomomzazsmbmo-
aslrstzoumtmsoonsomsmnomuoboszdnmnscoboscdooomcadlosmnzooxtoacbwmnmo-
ooorsomsunsamosoooouyzlonanmoczlmzoasznsyoommosltoosaunmocuobmstsmus-
omtozznanrawmsmmtnommyswonozoaooyuatrmlcsswntlawybbcazosmulutomnzoby-
zlouoomzzoloztnombooloosroltmmomwmwooalllsmotttsumamonlumotmmumsanlo-
mobsowymwssanmmbolzonoobnlolozmmaouoosorcbmnbmloaobnmozslmusysnsmcuo-
stsmrxoooslatzuoamwzmmdszwmooooooorzmmoxologoooolaolouulmasooomonwlmn-
osmzoouaounrnzoomonrltcdzouooylonazusonmbsluotmlonoanromlozomorzuocz-
sbormzounsmlzoaoodtommwuoacosacmawlulomotomztstsrbocamnuamrcawmsnmmm-
omszoomysaumasr

(word count)^2 / variance
10000 generations
100 chromsomes

papudvtehrarevrbtewedsmsfttepvbshrshuanehbphnteoephcdstdaisreptdmosu-
peyubyhusapvprmeenrdtywvhapoyhthynhrevsehhrevesneyvohsebehaspysnzdph-
umerteyhnpptftohvdtpvrueoesoztesarrmhderueayneraiwtytotbeyuutrwtygye-
rvnptowraaoeravnurevanreoatattwwmpvtutveswtrvytrwouvynwaaiorhtpbenea-

yranvuutopthavhevyuorwounhnaestyewyhewnvryeeaetbovtehonyvsobguuotaov-
verttwphnntpeteuopraurebyptttryyrahhpyotyeppobtbrgpthtvteooaoeupwypr-
woyoeyaptaortoneesatouvtpaooaeeybupeahonneaeooohaareenoaavoeaoevhpor-
eypevprubttrpeupuntepyvppnyodwmyhvotavepspewpybehvpneattoyahrtwuatov-
uotubiuevenhotoutabweewuwpauytryepoayrwvoaoyetdnhabeoooyrvdyaernboap-
vtatyrahuonbpwbaeenpnotwtvaryrnynhwaerpupeyodoeoepdtwbyyoyweyupworst-
eptnutuonventvhttwoaeyboyhsrtrsahboobryeodywruotpptthanohebshbaenaae-
rmvtaearvhoehvp

word count / variance
1000 generations
1000 chromosomes

hwzoqquiegjarvgjxtlpofdfsxrghqjweswzyndpgjrynurspreuqnrofdogafiqsako-
zahyjyeojnhgoysptejqurgvgwhshergbeggxowmygaaoapepytmgoajegffhbqekqhg-
ydugiergnohrsxmelqnfljyrsipmkiafwwgpyqugotchxaawtlzglewntryrlyglhprp-
ylkhqbuxkhepgcdhoxpoyhxtjwfcqrldpxoryfgaulinohhslmyqrhvgcmexgzfsthew-
gxnooyrqkxqjrqqoqjyjdvsyhghrourmtuegaghodgothulsbqiagjhhqujuproboyjo-
lwgxsveghfwnmlerbxarrdasgxisisnedkgzhrmusanxbswndpnhgrqqejmkborxuend-
vsgjuerrscmxlbhotpnimyqqfcmbhethsrnuwjmhkarobegqkygtihmkyqewhjtogxsa-
pgnadndrsolsxerfyksoeggkxkjmfgphgojicoofuntdfrstggfhtcrlmhkwdqdwrrkg-
oelrztyoqakgbsjywrsuepowvngygingxrvhhxtgmcagoifkjwsokjmhyofgnmxhsmhx-
qqrsgxkjyjhzxvuwtohokbrlrqkaedkghgvkaxxrfamkfbpmoefwdzgrhggtryruknur-
anxfrryjhotksggoilewprsknjuxosuhjsmjsxrogfhggwelffiughhjgpzujsrahrgt-
xposcphdogjegom

(word count)^2 / variance
1000 generations
1000 chromosomes

dnqzkvoaegmarxilzlqgtoboqusujvxsbsmtteaaifanxgrnaaenoostsgziogbkjftz-
qtnolubzmetjgcjaubfookkqznjnabsgjbiajnosjiiojtkbauyxilologsodjpbdojh-
tbribozixceeqzogfkfgsfjrnmwodwomvpgvjkrizuundoivvqxzfbpalyjrycjfxkha-
csdevpqugnggkuadzzgjuyzulnouvaqgvdjsooxtoqbftatsaoqlhtxrupgmixoqudon-
zmecjjrvpdvffkpzkkofbxnoygdkzoahuqegfjjjbizunrfapkboiljxkoltkrzpmulj-
byjfmxbbjopxsfzrpdafrbtqidwmbslebgikeyrbnoxdpapabkotbakvclrgpzrdqeob-
xnixreksmurzyptzuwebrjkvguciiijrraqzmnilannjizxactgeznhctsyziseabdcc-
mgfanqvastyvdiadzaujlckvdadikyirbssmcajhrfepvbhjbkdnecvyizcvviqznstk-
yyyagvajqnaiimgptnrsxtineqcpgpigtzfirpvkicmgjdkanmhjtdhrgrkghcpqrhid-
qittcpcnzsngdesmejneaisovqccjvagqbectmdavxutkitijlktqgybugyjopzstiuv-
xfegavasnaeamkbjnoymtgstynrmtvrinrighpojxkkybzyyhdpuxikgbtgrnrtnqnhe-
piatceivabsiboc