Breaking a Simple Substitution Cipher

Assignment 2 CS472-F16 Due: Mon Sep 26 at 5pm PT

Points: 200

Try to match the format and quality of answer in the results. You don't have to get an exact agreement. The comparison is with an example output. **Try to get similar amounts of error or less.**

Reference Files (these should be live links):

- rand.cpp A simple fast random number generator
- rand.h
- codes.tar Example encrypted text
- freq.txt English contact table
- pmx.cpp An example pmx xover operator

This is solving a non-order permutation problem with a GA.

This assignment we will try to break simple substitution ciphers and learn the basic format of a genetic algorithm. Allow plenty of time for this assignment.

The problem

The encoded message that is sent is called the cipher text. The unencoded message is called the clear text. The goal will be to write a GA to crack simple substitution ciphers and return the key and the clear text.

A simple substitution cipher is one in which each possible letter in the clear text has a unique translation into one character in cipher text. To encrypt the message one simply makes a one to one substitution of the letters in the clear text with the ones in the cipher text. The key is the mapping expressed as the alphabet of the clear text translated into cipher text alphabet. In our case this is the encoding for the 26 lowercase letters: "abc...z".

For example: The clear text:

```
"Programming is like sex: one mistake and you have to support it for
the rest of your life."
```

The key is the second line below

```
abcdefghijklmnopqrstuvwxyz
cpmtzkrhlsquniebdfaygwovjx     <---   the key
```

The cipher text broken up in the 5 letter blocks:

```
bferf cnnli rlaul qzazv eiznl aycqz citje ghcwz yeagb befyl
ykefy hzfza yekje gfulk z
```

In this problem we encode the message as follows. The clear text has had all uppercase letters changed to lowercase letters and all whitespace and punctuation removed. It is then enciphered with a substitution cipher determined by the key. Then the message is broken up in the traditional way of blocks of 5 letters each to hide the word breaks.

You will be given several enciphered texts each enciphered with a potentially different key. Your program must determine what the key and clear text is by evolving a key that optimizes a fitness function. Longer texts will be easier to solve than shorter ones. (Why?) Your algorithm should be able to crack the codes completely for longer

texts and only be off by a few infrequently used letters for shorter texts. You are not expected to get the perfect key in all cases. (Why?) You are not to attempt to recover the word breaks in a message, only the key.

Part 1:

Use the fitness function described in class (and below) and the contact frequency matrix supplied in the resources to formulate a fitness. You should write a GA of your choice (steady state or generational) and a permutation based xover and mutation of your choice. You may optionally perform a simple hill climbing local search if you think that will improve things. This can be done periodically or at the end as you choose. Experiment! You should not add in taboo or simulated annealing search since I would like to keep things simple and the problem will be solved without them. The goal here is not to solve these perfectly but to see how well they can be solved used these algorithms and what are the gotchas. Again, you should come within a few letters or perfect for the key and those should be infrequent letters like j, v, k. The longest messages you should miss by only a couple of infrequent letters worst case.

Part 2:

Augment the fitness function with a punishment function for bad choices as described in class and see if you can get improved performance in speed or quality. Be prepared to talk about your results in class.

The Fitness Function

To compute the fitness function for a key (a permutation of 26 letters) a **contact table** or **digraph table** for English is used. A contact table is a 26×26 matrix M where each element M_{ij} is the number of times each possible ordered pair of letters ij occurs in a sample of text i.e it is a count of occurrences. A contact table for English is supplied. It is derived from a large sample of English text.

The fitness of key K is derived by comparing the contact table for a large sample of English with the contact table from a message translated with the key K. Here is how the comparison is done: Let E be the contact table for English and C the contact table for cipher decoded by key K. Note that you only need to compute C once!!!

Let $e(i)$ be a function that encodes the letter i as defined by the key K. For the example key above $e(a) = c$ and $e(b) = p$.

Once the cipher text is read in you create the contact table for the cipher text. Since the counts in the contact table are just counts **they both need to be normalized**. Do this only once or your program will be too slow to finish. Let SUM_C be the sum of the counts in the C matrix and let SUM_E be the sum of the counts in the E matrix. These sums are used to normalize the matrices. The $fit(e)$ is the fitness of the encoding e. It is the square of the Euclidean distance. It is computed as the sum over all ordered character pairs ij as follows:

$$fit(e) = \sum_i \sum_j \left(E_{i,j} - C_{e(i),e(j)}\right)^2$$

where $e(i)$ and $e(j)$ are the encoding of the characters using the key K. E is the English digraph table and C is the digraph table from the encrypted text. NOTE: precompute what you can so you don't do hundreds of divides for ever fitness eval. IMPORTANT: The smaller the fitness the better the match (or just maximize the negative of this). As the key changes, the encoding changes and so does the fitness of the key.

An alternate and fairly successful fitness function is the Bhatthacaryya distance and is computed as:

$$fit(e) = \sum_i \sum_j \sqrt{E_{i,j} C_{e(i),e(j)}}$$

which must be maximized.

Consider what other rewards and punishments you can put into the fitness function. Discuss the plain fitness presented above and any others you might be interested in.

What to Turn In

You should turn in your code in a tar file along with a report. Your code and makefile should create a program named decipher . I will grade your program by compiling it and sending it files similar to the test files via standard input and see if it generates the key. The exact output format for the key is: `** yourSubmitName key` For example for the key above for user meerkat:

```
** meerkat cpmtzkrhlsquniebdfaygwovjx
```

This should be followed by a decoding of the message using the key.

On the lines that follow you can output whatever else you want as long as it does not begin with **. My code will be looking for the ** marker. There will be a time limit on execution of about 10 to 20 seconds. Be sure you write your fitness function efficiently!

You should report on your observations in a report called yourSubmitName.pdf . Put your name and assignment number at the top of the report! Thanks. Your report should describe your GA including the parameters such a xover used, xover probability, local search used if any, when it was used, etc There should be enough information that a reader could easily reconstruct your GA. Please put this in table form similar to what you see in the book. Then report on your successes in deciphering the eight sample texts. Explain what you tried that didn't work if any and why. Explain why you chose what you did. You can use additional pages for tables, diagrams, decipherments, keys, or plots but is not to present further explanations/observations.

Resources

Generic Two Letter Frequency Table for English

In the letter contact table, the first column is the letter pair. The second is the number of times it occurs in a large sample text. By summing up column two and dividing each entry in column two by that sum, you can get a normalized frequency. Not all character pairs are found in the table. Those that are not present are zero. e.g. "bf"

The Codes

For your testing pleasure, eight messages are provided each enciphered with a different key. They can be found in the files in codes.tar. The files are named with a letter and the length of the message:

```
b115.code
d158.code
e237.code
n250.code
m369.code
i603.code
o715.code
p1209.code
```

Also note that since there is no understanding of English by your program you should be able to send it a long plain English text and it should find the key is abcdefghijklmnopqrstuvwxyz. This is a quick and easy test that your program is working.

1 The Experiments

Write a simple half page report named "yourSubmitName.pdf" describing what you observe. Turn in your code and the report in a tar. The tar should include a makefile that will make the program `decipher`.

2 Grading this Assignment

I will grade this based first be seeing that your code compiles, runs and returns a sensible answer. I will read the report to make sure it is clear and that you understand what happened when you ran the experiments. Your report needs to be concise and to the point. Supply data to support your observations. You can run your code on the test system then write your report and resubmit them together.

3 Submission

Homework will be submitted as an uncompressed tar file to the homework submission page linked from the main class page. You can submit as many times as you like. The LAST file you submit BEFORE the deadline will be the one graded. For all submissions you will receive email giving you some automated feedback on the unpacking and compiling and running of code and possibly some other things that can be autotested. I will read the results of the runs and the reports you submit.

Have fun.