Algorithmics
S. Omran et al. "Using Genetic Algorithm To Break A Mono - Alphabetic Substitution Cipher" article review

Ilya Kuzovkin

May 23, 2012

1 Motivation

Heuristic algorithms in the field of operational research proved themselves to be quite successful in the attempts to find approximated solutions to various NP problems. One field of computer science, which deals with NP problems a lot, is cryptology. Almost all cryptographic primitives work under assumptions that some mathematical problem is in NP, and, therefore, is not solvable in reasonable time. The main question to ask here is: are these primitives still secure if we will try to attack them using heuristics? This article does not answer this question, but shows how genetic algorithm approach can be applied to cryptanalysis task, which does not have clear solution except for exhaustive search: breaking the substitution cipher.

2 Task description

Substitution cipher is very old and simple idea of creating encrypted message. Key to the cipher is one of the permutations of the alphabet. Example (spaces are ignored, case does not matter):

```
Alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Key:      PNQLRBVEKUOSIZJHYCGMTFDAWX

Plain:  THISISANESSAYFORALGORITHMICSCOURSE ("This is an essay for algorithmics course")
Cipher: MEKGKGPZRGGPWBJCPSVJCKMEIKQGQJTCGR
```

Breaking such cipher can be done easily by using letter frequency analysis. However, despite the fact that it is easy for human, it is not so obvious how the machine should solve this problem efficiently. Human solve the problem by locating two of three most frequent letters and, after their correspondences in the key are found, deducing other letters from common words like "the", "in", etc. Computer, however, cannot be easy programmed to use same deduction logic. Alternative way for computers is to go though all 26! possible keys, but this is not efficient.

3 Idea of the solution

Idea of genetic algorithm is to create a random population of solutions which will evolve to the correct solution over time. Evolution consists of following sequential steps:

1. Create the population.
2. Evaluate how good each population member is (using some objective function).
3. Choose some amount of best individuals.

4. Choose some additional amount of individuals for further breeding.
5. Mate.
6. Mutate.
7. Population size is now restored, go to Step 2.

In this context the proposed algorithm looks like this:

1. Generate population of size n (26-character strings) and fix number of generations g.
2. **For** 1 **to** g **do:**
   (a) Decrypt the cipher text by the n generated keys.
   (b) Evaluate each individual's efficiency on the objective function.
   (c) Sort the individuals (keys) based on the efficiency.
   (d) Keep $best = 20\%$ of best individuals for next generation.
   (e) Stochastically select $parents = 30\%$ among others to be parents for the next generation.
   (f) **For** 1 **to** $best + parents$ pairs **do:**
      i. Apply crossover to get *children* (50% pairs).
      ii. Apply mutation of 0.02% to the *children*.
      iii. Replace unused individuals with *children* to get a new population (n keys).
   (g) Go to Step 2.
3. Output is the best solution.

4 Implementation details

There are four pieces we need to define in order to be able to implement the algorithm.

4.1 Objective function

$$\text{score}_{key} = \alpha \sum_{i \in A} |K_i^u - D_i^u| + \beta \sum_{i \in A} |K_i^b - D_i^b| + \gamma \sum_{i \in A} |K_i^t - D_i^t|$$

where:

- A is alphabet.
- K and D denote letter frequency distributions in decrypted message and actual language respectively.
- α, β, γ are some coefficients in case we want to use these part with different weights.
- u, b, t stand for *unigram*, *bigram*, *trigram* (if we choose to use only unigrams, then parameters β ang γ can be set to 0).

4.2 Stochastic selection

All selection candidates are placed on a spin wheel, where each of them occupies sector size according to the score. We randomly pick starting point and then select required number of individuals by advancing $\frac{n}{360}$ degrees at each step. In such way probability to select weaker individuals is small.

4.3 Crossover

Notion of *crossover* is widely used in terms of genetic algorithms. It is the way how children are created from parental individuals. In our case it will be:

1. Choose two random parents
2. For each choose random crossover point
3. Split parent at crossover point
4. Swap second splits
5. Replace repeating elements with blanks
6. Insert missing elements into blanks
7. Two children are produced as a result

4.4 Mutation

Mutation is needed to prevent algorithm from being stuck at a local minimum. In our case the process is as follows

1. Generate random bit string of length $|A|$
2. Child is composed from 2 parts
   (a) First parent elements which correspond to 1 in the bit string.
   (b) Missing elements are inserted in the order they appear in second parent.
3. Second child is mutated in analogous way.

5 Results

Depending on the various parameters, number of discovered key letters varies from 8 to 17 (out of 26). It is promised that all letters can be opened as well, but this will require larger cipher texts to analyze. The result is considered successful, because from information given by the algorithm finding other letters is trivial. In strictly cryptological sense retrieval of any amount of true information about the key is considered as success.