EECS 4080 A Fall 2018 Final Report

Yunge Hao

Theory

Evolutionary Algorithms have been applied in searching for Boolean functions that have desired cryptographic properties in light weight Cryptography problems. Existing work shows that among other techniques, Genetic Programming yields one of the strongest results. I was focused on searching balanced, highly non-linear, and sufficiently correlation immune Boolean functions using Genetic Programming.

Basic setup

My project is based on ECJ System (Evolutionary Computation in Java). ECJ provides many handy libraries, user manuals and tutorials so the setup is simple enough for beginners. Everything is in one package. This package includes a class that defines the problem, the function classes, and a data type class. Before each searching starts, researchers should manually set the parameters in Koza.params, simple.params and the parameter file for the specific problem. In this project, there are two methods in Cryptograhy.java, setup and evaluate. The setup can be done by default. It is the evaluate part that determines the results.

Results

1. Balancedness

I tested inputs with sizes ranging from 4 to 10, and all the searches succeeded in the 0th generation. My conclusion is that random search by itself is enough to guarantee balancedness.

2. Non-linearity

For non-linearity of input sizes ranging from 4 to 10, I can find bent functions only when the number of inputs is even. They all succeeded in the 0th generation. No paper mentioned theoretical upper bound of non-linearity for odd number of inputs.

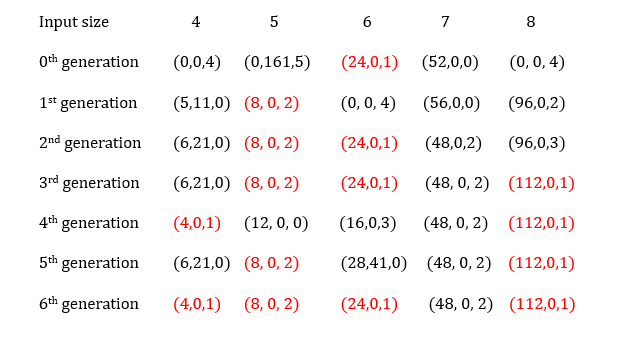
3. Balancedness and non-linearity

When searching for Boolean functions with high non-linearity as well as balanced, for inputs size from 4 to 8, no bent function was found. This is consistent with the literature. For input size 8, the best known of highest non-linearity when balanced is 116, and my result was 112. This number is not too far off, and it appeared still before the first generation, but it remained unchanged as far as 10 generations during my experiment. The search seemed to be stuck on some local optimal.

4. Balancedness, non-linearity and correlation immunity

When searching for three objectives together, balancedness, non-linearity and correlation immunity, the “good” functions appear not so often until more generations of evolution happened. By “good” I mean balanced, highly non-linear and correlation immune at least of degree 1. It also showed significant fluctuations through the evolution span, which had never been observed in the three previous cases.

In summary, the trace of gradual convergence in the history of evolution became clearer as the difficultness of finding desired result increased.



Results for 3 objectives: (non-linearity, balancedness, correlation immunity), red color indicates an optimal or a close hit.

The results for non-linearity has no improvements, i.e., it was still sub-optimal, as I expected, since involving correlation immunity made the search harder. The best individuals, however, did not degrade, compared with the previous results. It just took more time to find.

I tried several times on input size 10, the best I got was non-linearity 448, balanced, with correlation immunity of degree 3.

Future work and suggestions

1. I have not got the time to complete the experiment adding resiliency, which is another important property of Boolean functions. Resiliency of degree t will for sure tighten the upper bound of non-linearity as the papers claimed. I think this is a suitable starting point for the next student to carry on. It is easy since we already have the C code for calculating resiliency.

2. I suggest future researchers or students to try figuring out a way to let the search jump out of the local optimal, say, when detecting no change of results for some time, or a number of generations, stop the search and restart over from somewhere else. So keeping a memory of already searched spaces is necessary.

3. The Boolean function itself have not been fully explored. Perhaps there are some hidden patterns not revealed by the known properties of Boolean function. I was hoping to make use of such patterns so the running time can be reduced by eliminating duplicates in the data. I ended up finding none, however. This perspective is rather hard for non-mathematic major students. But it is triable.

4. There are many other fitness functions in ECJ’s library, they could be tired by future students on giving stronger results. The choice of fitness function determines how far the research can go. For example, when I was using fitness function that has only one objective, the best result for non-linearity of input size 8 is 64. Once I changed it to multi-objective, I got 112. How to perform selection may also decides whether the search would easily get stuck on a sub-optimal path or not. ECJ developers did not make it public how fitness functions are implemented. Indeed a researcher can define his/her own fitness function that fits this certain cryptography problem, but, again, it requires a quite deep mathematical background.

5. Maybe we can make this research project larger in scale, for example, a cross-major teamwork, by letting students and professors from math department work together with EECS students and professors.