International Journal of Computational Intelligence Systems, Vol. 4, No. 5 (September, 2011), 921-928

On the Application of Evolutionary Computation Techniques in Designing Stream Cipher Systems

Wasan Shaker Awad
Department of Information Systems, College of Information Technology,
University of Bahrain, Sakheer, Bahrain
Wasan_shaker@itc.uob.bh

Received 9 April 2010
Accepted 11 September 2011

Abstract

Encryption is an important mechanism used to protect private information from unauthorized access. Thus, cipher systems play an important role in the communication and storage systems. But, designing a cipher system of good properties, such as high degree of security and efficiency, is a complex process. Therefore, this paper considers this problem, and presents an attempt to find a general approach for designing good cipher systems automatically. In this paper we focus on an important class of cipher systems which is stream ciphers. The proposed approach is based on the evolutionary computation techniques, and the method chosen here is the simulated annealing programming which is an integration of genetic programming and simulated annealing algorithm. The proposed algorithm has been implemented in order to test its effectiveness in solving the underlying problem.

Keywords: Genetic Programming; Simulated Annealing; Stream Ciphers; Automated System Design.

1. Introduction

The explosive growth in computer systems and their interconnections via network has increased the dependence of organizations on the information stored and communicated using these systems. This, in turn, has led to a heightened awareness of the need to protect data. Therefore, a lot of work has been done information security and cryptography, which is the science of protecting private information against unauthorized access by encrypting it.

Any cryptographic system (cryptosystem, or cipher system) has five elements: plaintext (clear text), ciphertext (encrypted text), encryption algorithm which is a procedure used to encipher (encrypt) the plaintext and transform it to ciphertext, decryption algorithm which is the inverse of the encryption algorithm, and the key which is a parameter used to prevent the plaintext from being easily revealed by an authorized person.

Nowadays, you can find many cipher systems (cryptosystems), used for encrypting the private information, of different types, and a number of cipher systems have been proposed which are of different levels of security; these cryptosystems can be classified into modern and classical systems. There are a number of classical cipher systems, which are older than modern systems, such as Ceasar, Playfair, and Hill systems. Modern cipher systems are subdivided into block ciphers and stream ciphers. Block ciphers divide the plaintext into blocks and encipher each block independently, such as Data Encryption Standard (DES) and Advanced Encryption Standards (AES) systems, which are iterated product systems combining substitution and transposition [1,2]. Stream ciphers are extremely fast and easy to implement. In addition, they usually have very minimal hardware resource requirements. Therefore stream ciphers are of great importance in applications where encryption speed is paramount and where area-constrained or memory constrained devices make it impractical to use block ciphers. Stream ciphers can operate one data unit as small as a bit or a character.

Designing good stream ciphers is a complex process. Therefore, this problem has been considered in this paper. Of course, the only option we have for solving this problem is the heuristic techniques. Thus, the main

purpose of this paper is to present an automated general approach for designing stream ciphers that satisfy the desired properties. The proposed approach is based on the evolutionary computation techniques. The general algorithm will be presented in this paper, in addition to a detailed algorithm which is based Simulated Annealing and Genetic Programming. This algorithm has been implemented to investigate the effectiveness of evolutionary computation techniques in designing stream ciphers automatically. So far as we know, no such applications exist until now. Thus, this paper presents an initial effort on the application of evolutionary computation algorithms to design stream ciphers.

The problem considered here is the design automation of cipher systems. This problem has been considered by a number of researchers, In Ref. [3], Genetic Algorithm (GA) is used to find a set of rules of Cellular Automata (CA) suitable for cryptographic purposes. In Ref. [4] GA is used for the construction of Boolean functions for cipher systems, such as block ciphers and stream ciphers. The design of Boolean functions with properties of cryptographic significance is a hard task. Therefore, this problem has attracted a number of researchers, such as Millan, Clark, and Dawson [5], they have proposed a GA-based method for finding Boolean functions which are nearly always have high degree of nonlinearity. However, this problem has been reviewed in more details by Awad[6].

2. Stream Cipher Systems

Every stream cryptosystem consist of two parts, which are:

- Keystream (random sequence) generator, and
- Mixer (XOR for the binary sequence).

A keystream generator, which is the heart of stream ciphers, outputs a stream of bits (keystream) xored with a stream of plaintext bits to produce the stream of ciphertext, as shown in Fig. 1.

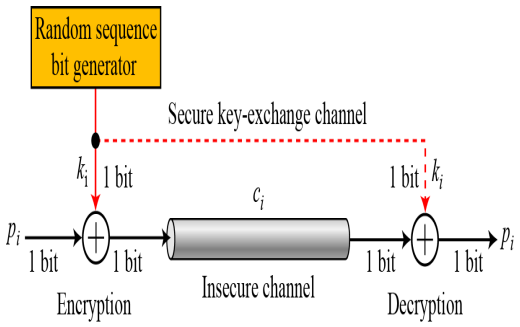

Fig. 1. Stream Cipher System

There are many stream ciphers which are currently widely used in our day life. Mainly, stream cipher systems can be classified into:

- Linear Feedback Shift Register (LFSR) based stream ciphers, in which a LFSR or nonlinear combination of LFSRS is used as keystream generator. Fig. 2 presents a LFSR [1,2,7,8,9].
- Nonlinear LFSR (NLFSR), in which a nonlinear feedback function is used [1,2,7,8,9].
- Feedback-with-Carry Shift Register (FCSR) [10].
- (n.k)-NLFSR [11]
- Cellular Automata (CA) [3].
- Algebraic Shift Register [12].

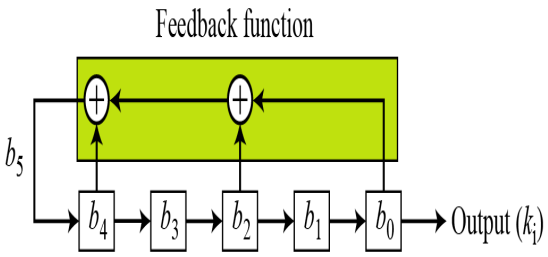

Fig. 2. LFSR

The stream cipher system's security depends entirely on the inside of keystram generator. The security of this generator can be analyzed in terms of randomness, linear complexity, and correlation immunity[13,14,15]. Thus, good stream cryptosystems must have the following features:

- They generate long period keystreams.
- Their keystreams are random.
- Large linear complexity.
- They have high degrees of correlation immunity.

A binary sequence is said to be random if there is no obvious relationship between the individual bits of the sequence. Several research efforts exist in the literature for developing suites of tests for evaluating random number (Binary keystream) generators to be involved in stream ciphers [13,14,15]. In all these methodologies two criteria are used for the evaluation of the quality of random numbers obtained by using some generator in traditional applications such as simulation studies: uniform distribution and independence. The most important requirement imposed on random number generators is their capability to produce random numbers uniformly distributed in [0,1]; otherwise the application's results may be completely invalid. A number of statistical tests are applied to examine

whether the pseudorandom number sequences are sufficiently random or not, which are frequency test, serial test, poker test, autocorrelation test and runs test.

i. **Frequency Test:** It calculates the number of ones and zeroes of the binary sequence and checks if there is no large difference.
ii. **Serial Test:** The transition characteristics of a sequence such as the number 00, 01, 10 and 11 are evaluated. Ideally, it should be uniformly distributed within the sequence.
iii. **Poker Test:** A N length sequence is segmented into blocks of M bits and the total number of segments is N/M. Within each segment, the integer value can vary from 0 to $m = 2M-1$. The objective of this test is to count the frequency of occurrence of each M length segment. Ideally, all the frequency of occurrences should be equal.
iv. **Runs Test:** A sequence is divided into contiguous stream of 1's that is referred as blocks and contiguous stream of 0's that is referred as gaps. If r_0^i is the number of gaps of length i, then half of the gaps will have length 1 bit, a quarter with length 2 bits, and an eighth with length 3 bits. If r_1^i is the number of blocks of length i, then the distribution of blocks is similar to the number of gaps.

Linear complexity is a well-known complexity measure in the theory of stream ciphers. Linear complexity of a keystream s is the length of the shortest LFSR which will produce the stream s, which is denoted by L(s). If the value of L(s) is L, then 2L consecutive bits can be used to reconstruct the whole sequence. Hence, to avoid the keystream reconstruction, the value of L should be large [16].

In order to obtain high linear complexity, several sequences can be combined in some nonlinear manner. The danger here is that one or more of the internal output sequences can be correlated with the combined keystream and attacked using linear algebra. A keysream generator has a higher degree of correlation immunity if there is no correlation between any internal output sequence and the combined keytream.

3. Genetic Programming and Simulated Annealing

One of the component methodologies of computational intelligence is evolutionary computation. Evolution is an optimization process, where the aim is to improve the ability of a system to survive in dynamically changing environment. There are number of evolutionary computation techniques, such as GA, Genetic Programming (GP), Cultured Algorithms, and Differential Evolution algorithms. Regardless of the technique used, evolutionary computation applications follow a similar procedure [17]:

a. Initialize the population.
b. Evaluate each individual in the population.
c. Select individuals.
d. Produce a new population by applying a number of operations on selected individuals.
e. loop to step *b* until some condition is met.

As mentioned above, there are number of evolutionary computation methods that can be applied to solve the problem of designing stream cipher systems. The proposed design algorithm is based on the integration of simulated annealing (SA) and GP which is called simulated annealing programming (SAP).

GP receives a high level statement of a problem's requirements from the user and attempts to create a computer program that provides a solution for the problem. In this paper, the computer program to be created represents a keystream generator. GP is the extension of the genetic model of learning the space of programs. These programs are expressed as trees. GP invented by John R. Koza in 1990s [18] which is regarded as an extension of GA [19, 20] attributed to John H. Holland [21]. Both techniques are identical in nature except for representation of individuals which in case of GP is parse trees based computer programs compared to fixed or variable length character strings in genetic algorithms. Representation is a major difference not only because it distinguishes the two techniques from each other but also because it greatly extends the problem handling capabilities of GP. It is one of the most promising domains independent and object oriented evolutionary computation techniques [22, 23, 24]. GP is used mainly for design automation and automatic programming; such as the design of analog and digital circuits [24].

On the other hand, SA, which has been introduced by Kirkpatrik et al [25], is a general randomization technique for solving optimization problems; it is a recent technique for finding good solutions to a wide variety of combinatorial optimization problems. This technique can help to avoid the problem of getting stuck in a local minimum and to lead towards the globally optimum solution. It is inspired by the annealing process in metallurgy. At high temperatures, the molecules of liquid move freely with respect to one another. If the liquid is cooled slowly, thermal mobility is lost. In SA, the solution starts with a high temperature, and a sequence of trail vectors are generated until inner thermal equilibrium is reached. Once the thermal equilibrium is reached at a particular temperature, the temperature is reduced and a new sequence of moves will start. This process is continued

until a sufficiently low temperature is reached, at which no further improvement in the objective function can be achieved. SA can be formulated as a graph with an energy E assigned to each node. In SA parlance, the nodes are called `states', the arcs represent `moves' from one state to a neighboring state, and the energy is sometimes called `cost'. Thus, SA algorithm consists of: configurations, re-configuration technique, cost function, and cooling schedule [26, 27].

Many researchers explored the application of SA on many different types of problems. Furthermore, SA can be integrated with GA or GP in order to work on a population of individuals and to preserve good individuals into the next generation [28, 29, 30, 31, 32].

4. The Proposed Method

The problem under study can be formulated as follows:

Given : Plaintext length in bits, which is the keystream length *size*.

Output : A LSFR-based keystream generator, that generate pseudorandom Binary sequence (keystream) of length size, and passes the randomness tests, such as frequency, serial, and run test.

We can apply the evolutionary computation general algorithm as a general method for designing keystream generators of the desired properties, as follows:

```
Algorithm: Keystream Generator Design
Input: Keystream Length
Output: Keystream Generator

Generate the initial population S1 randomly, the
individual members xi,i =0..population size, of the
population are candidate keystream generators;
Repeat
  Execute each xi in S1, to generate the Binary
   sequence of a given length;
  Measure the fitness value of xi;
  Adapt the current population S1 to produce S2;
  S1 ß S2;
Until (Maximum Number of generations);
```

In order to represent the candidate keystream generators, two representation methods can be used:

i. Binary string representation to represent the feedback function (linear or nonlinear) of the shift registers, and the Boolean functions which are used to combine a number of shift registers. Variable length chromosomes can be used.

ii. Executable structures (computer programs), in which the primitive functions are Boolean functions, and shift registers as example.

The fitness of an individual population member is measured by executing it to generate the keystream and then apply different tests, such as statistical tests, to measure the goodness of the keystream. These tests can be compiled in the fitness function.

In this work, SAP has been applied to solve the problem of designing stream ciphers (or more specifically keystream generators). Automated design is an essential part of GP paradigm.

The major steps for preparing GP for an application are [18]:

- Determining the function library.
- Determining the representation scheme.
- Determining the fitness measure.

The description of these steps is given in the following sub sections along the proposed algorithm parameters.

4.1. Function Library

In GP, the structure under adaptation is a set of programs representing the candidate keystream generators. The keystream generators considered here are LFSR-based generators. Thus, the important basic function which is the shift register should be included. The function library used in this work is presented in table 1. The proposed function library is sufficient since:

- It includes the LFSR function (SR), and there is no need to include other types of shift registers because for every shift register there is an equivalent LFSR.
- Any combinational logical function can be expressed using (AND) and (XOR) only, that is because, any logical function can constructed from (AND), (OR), (NOT), and

NOT x = x XOR 1 (1)
x OR y = NOT (NOT(x) AND NOT (y)) (2)

4.2. Representation Scheme

The population chromosomes (programs), that represent candidate keystream generators, are strings of characters which are expressions represented using prefix polish notation. Fig. 3 shows the syntax of the population programs. These syntactic rules should be preserved during the generation of the initial population, and by

the genetic operations. Therefore, strongly-typed GP [33] is used.

The initial states and feedback functions of the shift registers are represented as strings of the letters 'a'..'p'. These letters represent the numbers 0..15. Thus, each letter is a sequence of four bits. The length of a LFSR is determined by the number of letters which initially are generated randomly. The number of these letters must be even, half of them for the initial state, and the second half for the feedback function.

For example, if the number of these letters is 8 letters, then the length of LFSR is 16 bits (4 x 4). Furthermore, the first zeros of the feedback function are ignored. For example, consider the following LFSR: "> abid", 'i' is the number 8 = $(0001)_2$, then the first three zeros are ignored, and the length of this LFSR will be five bits (1 + 4). Thus the feedback function will be (11100).

Table 1. The function library used

Symbols	Arty	Format	Description
SR	2	SR x	Shift register where x represents the feedback polynomial and initial state.
&	2	& x y	Bitwise AND operation between the two binary sequences x and y.
^	2	^ x y	Bitwise XOR operation between the two binary sequences x and y.
X	0		Sequence of characters 'a'..'p', representing numbers 0..15.
|	2	| x y	Bitwise OR operation between the two binary sequences x and y.

S à SR X | & S S | ^ S S | | S S

X à aX | bX | ... | pX | a | b | ... | p

Fig. 3. The syntax rules of GP language.

4.3. Fitness Function

The fitness value is a measurement of the goodness of the keystream generator, and it is used to control the application of the operations that modify a population. There are a number of metrics used to analyze keystream generators, which are keystream randomness, linear complexity and correlation immunity. Therefore, these metrics should be taken in our account in designing keystream generators, and they are in general hard to be achieved.

The fitness value is calculated by generating the keystream after executing the program, and then the generated keystream is examined. The fitness function used to evaluate the chromosomes is to calculate at what percentage the chromosome satisfies the desired properties of the stream ciphers. For example, Eq. 3 is fitness function which is used to evaluate of the chromosome based on the keystream randomness tests: frequency and serial tests. In which *nw* is the frequency of *w* in the generated binary sequence, and *size* is the keystream length. This fitness function is derived from the fact that in the random sequence:

- Probability (no) = Probability (n1), and
- Probability (n01) =
  Probability (n11) =
  Probability (n10) =
  Probability (n00)

$$fit(x) = \frac{size}{1 + |n_0 - n_1| + |n_{00} - size/4| + |n_{01} - size/4| + |n_{10} - size/4| + |n_{11} - size/4|} \quad (3)$$

There is another randomness requirement which is $1/2^i$* nr runs in the sequence are of length *i*, where *nr* is the number of runs in the sequence, *M* is maximum run length, and n_i is the desired number of runs of length *i*. Thus:

$$\left(\frac{1}{2^i} \times n_r \right) = n_i \quad forall \quad i = 1 \; to \; M \quad (4)$$

This can be expressed in the fitness function as follows:

$$f = \sum_{i=1}^{M} \left| \left(\frac{1}{2^i} \times n_r \right) - n_i \right| \quad (5)$$

Then the fitness function can be modified to be as follows:

$$fit(x) = \frac{size}{1 + |n_0 - n_1| + |n_{00} - size/4| + |n_{01} - size/4| + |n_{10} - size/4| + |n_{11} - size/4| + f} \quad (6)$$

4.4. The Algorithm Parameters

The genetic operations used to update the population are 1-point crossover with probability 1.0 and mutation with probability 0.05. The selection strategy, used to select chromosomes for the genetic operations, is the 2-tournament selection. The initial value of the SA temperature is 250 which is updated by multiplying it by 0.95. The decision of using these parameters values was made based on the results of a number of

experiments that show that with these values the proposed algorithm performance is the best.

The old population is completely replaced by the new population which is generated from the old population by applying the genetic operations.

Regarding the structure of each chromosome, the maximum chromosome length is 300 characters, and the maximum number of functions (except SR) is ten functions. The probability of the function SR is 0.5, and all other function are of probability 0.5. Finally, the maximum LFSR length is 20 bits.

The run of GP is stopped after a fixed number of generations. The solution is the best chromosome in the last generation.

4.5. The Design Algorithm

The proposed algorithm for designing a keystream generator that meets the desired properties is based on the integration of SA and GP, i.e., simulated annealing programming. The reason behind this integration is that, the performance of the algorithm was improved with the use of SA. That is because SA can help to avoid the problem of getting stuck in a local minimum, and to preserve good individuals into the next generation.

The following is the complete algorithm:

```
Algorithm:  Simulated Annealing Programming
Input : Keystream length (size)
Output : LFSR-based keystream generator

Generate the initial population (pop) randomly;
Evaluate pop;
Temp ß 250.0;  //temperature
Repeat
 Generate a new population (pop1) by applying
   crossover and mutation;
 Evaluate the fitness of the new generated
  chromosomes of pop1;
 Calculate the averages of fitness values for pop and
 pop1, av1 and av2 respectively;
 If (av2 > av1)  then  replace the old population by the
   new one, i.e. pop ß  pop1;
 Else
 Begin
       e ß  av1-av2;
       Pr ß  e / Temp;
       Generate a random number (rnd);
       If (exp(-pr) > rnd)  then  pop ß  pop1;
 End;
 Temp ß  Temp * 0.95;
Until (Max Number of generations);
```

As shown in the algorithm, SA is the technique used for the construction of the keystream generators. The structure under adaption is the set of GP expressions, and the GA operations are used to update the population of expressions.

5. Results

The main purpose of this paper is to design an effective algorithm for designing keystream generators, and also to study the effectiveness of GP and SA to solve this problem. Therefore, the proposed method has been implemented using C++ programming language. The proposed method has been applied to design keystream generators of different values of keystream length (size), and population size. The results are presented in tables 2, 3, and 4. These results are derived from running the algorithm 100 times.

Also, to make a comparison between simple GP and SAP, GP has been applied. Fig. 4 presents the results as the average of best fitness values in 100 runs for different values of pop. Size (100-550) and the keaystream length is 200 bits. It is clear that the performance is highly improved by applying SAP.

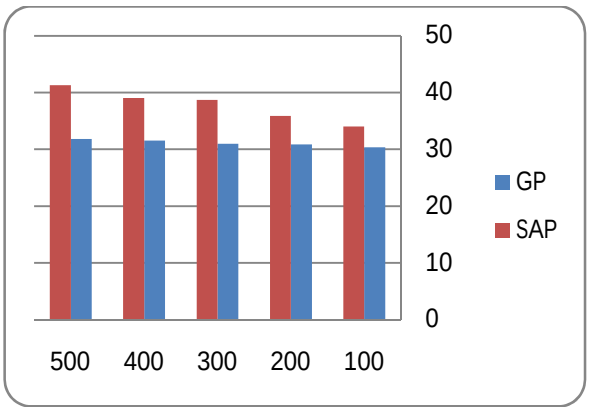

Fig. 4. Comparison between GP and SAP

Table 2. Best chromosome (solution)

Keystream Length (size)	200	100	50
Pop. Size	Fitness value Keystream generator		
100	48.3019 SR hgjj	26.5285 SR almj	11.0535 SR haih
200	48.3019 SR kehe	26.5285 SR cnkc	11.1986 SR jekdjfge
300	48.3019 SR dhjj	26.5285 SR cfkc	11.1986 SR jekdjfge
400	48.3019 SR cddd	26.5285 SR jkpl	11.1986 SR cmdlljbp

500	48.5308 SR bjpoa	26.5285 SR cfjf	11.1986 SR chbhik

Table 3. Average of best fitness values

Keystream Length (size)	200	100	50
Pop. Size		Average	
100	34.01549	21.8774534	10.008671
200	35.87525	24.436832	10.313523
300	38.72688	24.972075	10.503324
400	39.03217	25.401894	10.601804
500	41.31128	25.726553	10.800264

Table 4. Frequency of the best chromosome

Keystream Length (size)	200	100	50
Pop. Size		Frequency	
100	3	5	0
200	9	15	8
300	11	21	9
400	26	30	10
500	23	39	34

According to the results, the population size and keystream length affect the effectiveness of the proposed method. Table 2 shows that the fitness value of the best chromosome increases as population size increases, and longer keystream yields better solution. Also, we can see from table 2, all solutions are linear keystream generators. That is because; the linear complexity has not been taken in our consideration in the evaluation of the chromosomes. Table 3 presents the averages of the fitness values of the best chromosomes in 100 runs. It is clear that increasing the population size improves the effectiveness of the algorithm.

Example: The following chromosome that represents a keystream generator is of fitness value 50.66:

SRphikje

The following Binary keystream that is generated by this linear generator is of period length greater than 200 bits:

11111110000011011101000000000010101110000110101010
10110011110101110011001100010000011000010001000101
10101000101001011010011100110100110110001101111011
10010001110100011111000010010111100101111110101101

6. Conclusion

In this paper, a new approach for designing keystream generators automatically is presented, which is a new promising direction for stream cipher design. It has been shown the capability of GP and SA in designing the desired stream ciphers. Stream cipher design method presented here can be used for evolving any generator that satisfies the given requirements, such as period length, and randomness. These requirements are expressed mathematically in the fitness function. The experiments have shown the feasibility of the proposed method. Furthermore, the evolved keystream generators in the most cases are linear. However the results obtained can be improved by considering more factors in the fitness evaluation such as linear complexity, other randomness tests, and correlation immunity. Furthermore, other techniques and genetic operations can be examined and compared in order to find the best algorithm for solving the underlying problem.

References

1. B. Schneier, *Applied cryptography* (John Wiley and Sons, NY, 1996).
2. B. A. Forouzan, *Cryptography and network security* (McGRAW-HILL, NY, 2008).
3. M. Szaban, F. Seredynski, and P. Bouvry, Collective Behavior of Rules for Cellular Automata-Based Stream Ciphers, *IEEE Congress on Evolutionary Computation*, (July 2006), pp. 179-183.
4. Clark and L. J. Jacob, Almost Boolean functions: the design of Boolean functions by spectral inversion, *Computational Intelligence* 20 (3) (2004) 450-462.
5. W. Millan, A. Clark, and E. Dawson, An effective genetic algorithm for finding highly nonlinear Boolean functions, *in Proc.1st Int. Conf. on Information and Communications Security* (China, Beijing, 1997), pp. 149-158.
6. W. S. Awad, The applications of GA in cryptology, *Far East journal of experimental and theoretical artificial intelligence* 2 (1) (2008) 59-76.
7. S. W. Golomb, *Shift Register Sequences (*Holden-Day, San Francisco, 1967).
8. Beker and F. Piper, *Cipher Systems* (John Wiley, NY, 1982).
9. R. A. Rueppel, *Analysis and Design of Stream Cipher* (Springer-Verlag, NY, 1986).
10. Elena Dubrova, Maxim Teslenko, and Hannu Tenhunen, Analysis and Synthesis of (n,k)-Non-Linear Feedback Shift Registers, *in proc. of the conf. on design, automation and test,* (Munich, Germany, 2008), pp. 1286-1290.
11. Klapper and M. Goresky, Feedback shift registers, 2-adic span and combiners with memory, *Journal of Cryptology*, (10) (1997) 111–147.
12. M. Goresky and A. Klapper, Pseudonoise Sequence Based on Algebraic Feedback Shift Registers, *IEEE Trans. Inf. Theory*, 52 (4) (2006) 1649-1662.
13. H. Gustafson, et al, A computer package for measuring the strength of encryption algorithm, *Comp. & Sec.* 14 (1994) 687-697.

14. K. Zeng, C. Yang, and T.R.N. Rao, Pseudorandom Bit Generator in Stream Cipher Cryptography, *Comp*. 24 (2) (1991) 8-17.
15. L'ecuyer, P. and Simard, R, TestU01: A C library for empirical testing of random number generators, *ACM Trans. Math. Softw* 33 (4) (2007) 22-40.
16. Massey J. L., Shift register sequences and BCH decoding, *IEEE Transaction on Information Theory* IT-15 (1) (1976) 122-127.
17. Russell Eberhart & Yuhui Shi, *Computational Intelligence: concepts to implementation* (Morgan Kaufmann, San Fransisco, 2008).
18. J. R. Koza, *Genetic programming* (MIT press, Cambridge, 1992).
19. D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning* (Addison-Wesley, NY, 1989).
20. Melanie Mitchell, *An Introduction to Genetic Algorithms,* (MIT Press, Cambridge, 1996).
21. J. H. Holland, *Adaptive in natural and artificial systems* (Ann Arbor, University of Michigan, USA, 1975).
22. Haym Hirsh, Wolfgang Banzhaf, J. R. Koza, Conor Ryan, Lee Spector, and Christian Jacob, Genetic programming, *IEEE Intelligent Systems* 15 (3) (2000) 74-84.
23. J. R. Koza, M. A. Keane, and Matthew Streeter, What's AI done for me lately? - genetic programming's human competitive results, *IEEE Intelligent Systems* 18.(3) (2003) 25-31.
24. J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs* (MIT Press, Cambridge, 1994).
25. S. Kirkpatrik, et al., Optimization by simulated annealing, *Science* 220 (4598) (1983) 671-680.
26. Liu Yong, Kang Lishan, and D. J. Evans, The annealing evolution algorithm as function optimizer, *Parallel Computing* 21 (3) (1995) 389-400.
27. Van Laarhoven, P. J. M., et al., *Simulated Annealing: Theory and applications* (Reidel, Holland, 1987).
28. Sadegheih, Sequence optimization and design of allocation using GA and SA, *Applied Mathematics and Computation,* 186 (2) (2007) 1723-1730.
29. U. Yuichiro, M. Mitsunori, H. Tomoyuki, Simulated Annealing Programming Using Effective Subtrees, *Doshisha Daigaku Rikogaku Kenkyu Hokoku* 49 (4) (2009) 205-209.
30. M. Miki, M. Hashimoto, Y. Fujita, Program Search with Simulated Annealing, *in proc. of the 9th annual conference on Genetic and evolutionary computation,* (London, England, 2007), pp. 1754 – 1754.
31. O. Cordon et al. An Inductive Query by Example Technique for Extended Boolean Queries Based on Simulated-Annealing Programming, in the proc. Of *7th International ISKO Conference on Challenges in Knowledge Representation and Organization for the 21st Century. Integration of Knowledge across Boundaries*, ed. M.J. López-Huertas *(Granada, Spain, 2002), pp. 429-436.*
32. Ali Pahlavani, A New Fuzzy MADM Approach and its Application to Project Selection Problem, *International journal of computational intelligence systems* (2010) 103 – 114.
33. T. Haynes, et al., Strongly typed GP in evolving cooperation strategies, *in Proc. of the sixth Int. Conf. on GA*, ed. Eshelman, L.J. (Morgan Kaufmann, July 1995), pp. 271-278.