Proceedings of the 7th WSEAS International Conference on Evolutionary Computing, Cavtat, Croatia, June 12-14, 2006 (pp20-27)

Experiments on Cryptanalysing Block Ciphers via Evolutionary Computation Paradigms

Nalini N
Department of Computer Science and Engineering,
Siddaganga Institute of Technology, Tumkur 572103, Karnataka, India

Raghavendra Rao G
National Institute of Engineering, Mysore 570008, Karnataka, India

Abstract: Evolutionary computation techniques have been effectively used to solve a number of optimisation problems. Cryptanalysis of ciphers has been attempted through several methods such as brute force attack, linear and differential cryptanalysis and heuristic-based approaches. Most papers in the literature use heuristic search to classical ciphers such as simple substitution or transposition ciphers. To demonstrate the power of evolutionary techniques for attacks of modern-day ciphers, we report for the first time systematic experiments on heuristic-based attacks of Simplified Data Encryption Standard (SDES) and Modified version of Data Encryption Standard (DES) and present our conclusions. Though these are simpler ciphers, they contain the significant building blocks and features present in other complex ciphers. Thus the studies reported in this paper will be useful for the heuristic attack of other similar ciphers.

Keywords: Cryptanalysis,EvolutionaryComputation, Block ciphers, GeneticAlgorithms, ParticleSwarm Optimisation,TabuSearch,Simplified Data Encryption Standard

1 INTRODUCTION

Automated cryptanalysis has gained significant interest in recent years since human interaction is a time-consuming process. In several cryptanalysis problems, from among a large number of potential solutions, the candidate keys not eligible should be eliminated in a systematic manner. Such a pruning of the search space is possible using combinatorial optimisation heuristics. Studies on cryptanalysis of even simple block ciphers using heuristic search methods are useful for attacks of more complex ciphers with similar characteristics/structure. An attack on a cipher can make use of the ciphertext alone or it can use some plaintext and its corresponding ciphertext (referred to as a known plaintext attack). The first category of attack is harder and more challenging and thus we consider such an attack in this paper. Automated techniques can also be useful in the area of cipher design.

Significant amount of research work has been reported on cryptanalysis of ciphers which consist of a combination of a number of simple operations such as substitutions and permutations. Clark et al.[1] have carried out interesting research on the use of optimization heuristics such as genetic algorithms (GA) [3],[9],tabu search [2] and simulated annealing [5], for the automated cryptanalysis of classical ciphers. These papers consider simple substitution and permutation ciphers. Cryptanalysis of a simple substitution cipher has been discussed by Spillman et al. [7]. Knapsack cipher attack is described by Spillman [6].

Realising that studies on the attack of practical complex cryptosystems using evolutionary techniques have not been reported, we present in this paper our study on the cryptanalysis of Simplified Data Encryption Standard (SDES) and a modified version of DES (Data Encryption Standard) designed to make the implementation effort tractable. To the best of our knowledge, such efforts have not been reported in the literature. Though these are comparatively simpler ciphers, such studies will give better insights into the attack of DES and other present day ciphers, using evolutionary

methods. Though some of these simplified ciphers are amenable to brute force attacks, studies reported in this paper are useful in the cryptanalysis of other complex ciphers and in exploring the weakness of ciphers. The basic building blocks of most block ciphers have certain similarities, it is thus felt that our studies on evolutionary computation can be extended to study attacks of other ciphers. Also if evolutionary computation-based approach is unsuccessful on these simple systems, it is unlikely to be worthwhile to apply such approach to more complicated systems. As candidate evolutionary computation techniques, we consider here genetic algorithm, and its adaptive variant, particle swarm optimisation and tabu search method.

In view of the above motivation to carry out this study on simpler ciphers, we consider the modified versions of DES with 16 bit key without S-box and 4 rounds; other than the SDES. The rest of the paper is organized as follows. Section 2 presents a brief overview of the ciphers considered in this paper. The evolutionary approaches considered in this research work are summarized in section 3. Section 4 presents the experimental studies carried out and the results obtained for the SDES algorithm. Experiments and the relevant results for modified DES are discussed in section 5. Conclusions are presented in section 6.

2 SIMPLE BLOCK CIPHERS STUDIED

2.1 The SDES Algorithm

The SDES [10] encryption algorithm takes an 8-bit block of plaintext and a 10-bit key as input and produces an 8-bit block of ciphertext as output. The decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used as input to produce the original 8-bit block of plaintext. The encryption algorithm involves five functions; an initial permutation (IP), a complex function called f_K which involves both permutation and substitution operations and depends on a key input; a simple permutation function that switches (SW) the two halves of the data; the function f_K again, and a permutation function that is the inverse of the initial permutation (IP^{-1}). The details of SDES algorithm are not presented here due to lack of space and can be found in [10].

2.2 Modified DES

IBM designed Data Encryption Standard in 1970; later it was adopted as a standard. A detailed description of the algorithm is provided in [10].

Simplifying the Problem

For the purpose of cryptanalysis, two cases were considered.

1. Reduced number of rounds

Here we considered only few rounds of DES algorithm so as to simplify the problem.

Typically in cryptanalysis we used DES with six rounds.

2. Removing SBOX

The strength of DES lies in the non-linearity induced by SBOX, so we thought of initially eliminating the SBOX in the encryption and decryption phases. But looking into the algorithm, the elimination of SBOX is not straightforward. It is clear that each round takes 32-bit inputs Li_{-1} and R_{i-1} from the previous round, and produces 32-bit outputs L_i and R_i, for $1 \le I \le 16$,as follows;

$$L_i=R_{i-1} \quad (1)$$

$$R_i=L_{i-1} \oplus f(R_{i-1}, Ki) \quad (2)$$

where $f(R_{i-1}, K_i) = P(S(E(R_{i-1}) \oplus K_i))$. (3)

From equation 3, R_{i-1} (32-bit) is expanded to 48-bit using Table 1 and it is XORed with K_i . Result of this is subjected to SBOX, to get a 32-bit output. To eliminate the SBOX, we use Table 2 after XOR operation in equation 3. We have derived the entries in the table from the expansion table shown in Table 1,by eliminating all those entries, which occur second time. The input to an SBOX is 6 bits and output is 4 bits. The above operations ensure the flow of this bit pattern when the SBOX is removed. Removal of the SBOX needs special consideration to develop the steps of the algorithm.

The Algorithm

The standard DES algorithm and its variant for the 16-bit case are presented below.

INPUT: Plaintext m1,…,m64; 64-bit key K=k1,…,k64 (including 8 parity bits)
OUTPUT: 64-bit cipher text block C=C1,…,C64
1.Key schedule: Compute sixteen 48-bit round keys Ki from K.

2. (L0, R0)←IP (m1m2…m64)
3.for I from 1 to 16,compute Li and Ri
a. Expand Ri-1=r1r2…r32 from 32 to 48 bits, T←E (Ri-1)
b. T'←T⊕ Ki.Represent T' as eight 6-bit character strings (B1…B8)=T'
c. T"←(S1 (B1), S2 (B2)…S8 (B8))
d. T"'←P (T")
4.b1b2…b64←(R16, L16)(Exchange final blocks L16, R16)
5.C←IP^{-1} (b1b2…b64)

16-bit key DES

We have designed 16-bit DES algorithm and used heuristics for cryptanalysis of this cipher. The details of the algorithm and the relevant tables are not shown here due to lack of space.

Table 1: Expansion and Permutation

E					
32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

P			
16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Table 2:Eleiminating SBOX

3 EVOLUTIONARY ALGORITHMS

3.1 Fitness Function or Cost

We consider various fitness functions based on the following features.
1. Monogram and bigram statistics

To implement this fitness function, the frequency of each character in the decrypted text is calculated. This frequency is normalized by dividing it by the total number of characters in the file. This normalized frequency is then subtracted from the expected frequency of the character in normal English text. The absolute value of this difference is taken. The differences for all characters are added together. The normalization takes care that this value always lies between 0 and 1.

The bigram is an extension of unigram to two characters. Now rather than calculating frequency of individual character, we calculate frequency of "pairs" of letters. For example, a pair "an" will always appear more frequently than pair "bt". Again statistics for the frequencies of these pairs are also available. These statistics are compared with the statistics obtained from the decrypted text.

To implement this fitness function, the frequency of each pair of letters in the decrypted text is calculated. This frequency is normalized by dividing it by the total number of pairs in the file. This normalized frequency is then subtracted from the expected frequency of the pair in normal English text. The absolute value of this difference is taken. The differences for all pairs are added together. The normalization takes care that this value always lies between 0 and 1.

The fitness function based on monogram and bigram is given by,

$$\sum\{|SF[i]-DF[i]|+\sum|SDF[i,j]-DDF[i,j]|\}/4,\ i=1 \text{ to } 26, j=1 \text{ to } 26. \quad (4)$$

Here the letters A…Z are referenced by the indices 1…26, SF [i] is the standard frequency of character i in English, DF [i] is the measured frequency of the character i in English. SDF is the standard bigram frequency and DDF is the decoded bigram frequency.
2.Count of intelligible characters represented by ASCII
This is the most simple, efficient and effective fitness function of all the fitness functions tried in our experiments. To calculate fitness of a key, we simply calculate the number of readable characters in the decrypted text(a-z and A-Z).

We further normalize this value by dividing this number by the size of the file in bytes. For a perfect key, all the characters will be readable and hence this value will be one. Since our algorithms are implemented to solve minimization problems, we simply subtract this value from 1 to get the fitness of a key.

Intuition behind this approach is that the message is made up of readable characters. So the decrypted text should also contain the readable characters.

More the intelligible characters, better is the key, lower is the fitness value of the key. Since this approach does not require any table lookups, it is also the most efficient approach seen so far.

In the experiments performed, it was found that this function does give fair weight to the key. The fitness value decreases more rapidly than other fitness function values. It also spans almost the complete range from 0 to 1.

The formula for this fitness function can be given as follows:

f = 1-c/n;

where c = number of characters falling in the range a-z or A-Z

n = total number of characters in the file.

3. Combination of different cost functions with different weights:

In one more approach of assigning the fitness values to individuals, we used combination of more than one fitness function and assigned weights to it. Thus the new fitness value was calculated as follows:

f=α*unigram+ β*bigram+μ*intelligible_char

where α, β, and μ are the weights of the respective fitness functions and α+β+μ=1. This fitness function was also found to be quite useful during our cryptanalysis studies.

3.2 Genetic Algorithm

Genetic algorithms are developed based on the idea of emulating the evolution of a species. Details on genetic algorithms and their application to optimisation problems are extensively treated by Goldberg [3] and Srinivas et al. [9]. Keys in cryptanalysis studies are represented as a string of bits in the chromosome and genetic operators process this bit string. The fitness function is given in eqn. (4).

3.3 Adaptive Genetic Algorithm

A Genetic Algorithm gets stuck in local minima. One of the most celebrated ways to make a genetic algorithm come out of local minima is Adaptive Genetic Algorithm [8]. We have seen that there are many parameters like probability of crossover, probability of mutation, etc. which control the execution of a Genetic Algorithm. Usually these parameters remain constant throughout the execution of the program. This is exactly where Adaptive Genetic Algorithm (henceforth AGA) differs from normal Genetic Algorithm (henceforth GA). As we shall see, this difference makes AGA [8] perform a lot better than GA in difficult conditions.

The significance of pc and pm, probabilities of crossover and mutation, in controlling GA performance has long been acknowledged in GA research [9]. The choice of *pc* and *pm* is known to critically affect the behaviour and performance of the GA, and a number of guidelines exist in the literature for choosing *pc* and *pm*. These generalized guidelines are inadequate as the choice of the optimal *pc* and *pm* becomes specific to the problem under consideration.

Most intuitive way of changing the values of pc and pm is to change them according to the fitness values of the individuals. Obviously, we want to give more chances for crossover for "good" parents. Also, "good" children should be refrained from being perturbed by mutation. For this reason, probability of crossover, pc, should be higher for parents having low fitness values (remember, we are dealing with minimization problem rather than a conventional maximization problem). At the same time, probability of mutation, pm, should be lower for children having low fitness values. Thus, in this *adaptive* technique, we will have different values of pc and pm for different generations. Having noted these proportionalities between fitness values and required values for pc and pm, we now discuss the exact formulae to calculate these values on the fly.

[8] Provides good guidelines for the calculation of pc and pm for each individual for a maximization problem. Since we are dealing with minimization problem, we have modified these formulae slightly to adjust them for our problem. The formulae are,

pc= k1 / (favg-fmin);

pm= k2 / (favg-fmin);

where, fmax = the fitness of the best key in the generation;

fi= fitness of the key under consideration.

Also k1 and k2 are the scaling parameters which are fixed before the execution of the program. As we can see, higher the value of favg, higher the difference (favg-fmin) and lower is the value of pc. Thus, lower values are given less chance of crossover. Values of k1 and k2 are fixed in such a way that values of pc and pm always remain in the range 0-1.

Thus having changed values of pc and pm, the problem of local minima is solved to some extent. AGA does come out of local minima providing improved results as compared to GA. However, a stage reaches when average population seizes to change. At this stage, AGA saturates and values of pc and pm remain constant. AGA then converges back to GA and gets stuck in local minima again. To solve this problem, we have come up with somewhat

haphazard but effective scheme of periodic changes in pc and pm values.

3.4 Tabu Search

The tabu search [2] prevents the search from returning to a previously explored region of the solution space too quickly. This is achieved by retaining a list of possible solutions that have been previously encountered. These solutions are called 'tabu'; hence the name of the technique. The size of the tabu list influences the performance of the algorithm. Tabu search is similar to simulated annealing with the added constraint of the tabu list. Two randomly chosen key elements are swapped to generate candidate solutions. In each iteration, the best new key formed replaces the worst existing one in the tabu list. The algorithm can be found in [2].

Though strictly not based on evolutionary paradigm, tabu search is considered in this work because of certain similarities it possesses with the genetic algorithm. The tabu search, like the genetic algorithm, introduces memory structure into its workings. The tabu search utilises memory to prevent the search from returning to a previously explored region of the solution space too quickly. In each iteration, a list of candidate solutions is proposed and these solutions are obtained in a similar fashion to the mutation operation used in the genetic algorithm.

3.5 Particle Swarm Optimization (PSO)

In [4] James Kennedy et al. propose a new technique called particle swarm, which borrows the idea from bird flocking, fish schooling, and swarm theory. We have used this technique in the cryptanalysis of DES.

3.5.1 The Principle

PSO shares many similarities with evolutionary computation techniques such as Genetic Algorithms (GA). The system is initialized with a population of random solutions and searches for optima by updating generations. However, unlike GA, PSO has no evolution operators such as crossover and mutation. In PSO, the potential solutions, called particles, fly through the problem space by following the current optimum particles. Compared to GA, the advantages of PSO are that PSO is easy to implement and there are few parameters to adjust. PSO has been successfully applied in many areas; function optimization, artificial neural network training and fuzzy system control. PSO has been used in cryptography to solve the tough problem of integer factorization.

PSO simulates the behavior of bird flocking. Suppose a group of birds are randomly searching food in an area. All the birds do not know where the food is. The effective strategy is to follow the bird which is nearest to the food. In PSO, each single solution is a "bird" in the search space. We call it "particle". All particles have fitness values, which are evaluated by the fitness function to be optimized, and have velocities which direct the flying of the particles. The particles fly through the problem space by following the current optimum particles.

PSO is initialised with a group of random particles (solutions) and then searches for optima by updating generations. In each iteration, each particle is updated by following two "best" values. The first one is the best solution (fitness) it has achieved so far. This value is called best. Another "best" value that is tracked by the particle swarm optimiser is the best value, obtained so far by any particle in the population. This best value is a global best and is called gbest. When a particle takes part of the population as its topological neighbours, the best value is a local best and is called lbest.

After finding the two best values, the particle updates its velocity and position with the following two equations:

v[]=v[] + p_incr * rand() * (pbest[] - present[]) + g_incr * rand() * (gbest[] - present[]) (6)

present[] = persent[] + v[] (7)

v[] is the particle velocity, persent[] is the current particle (solution), pbest[] and gbest[] are defined as stated before, rand () is a random number between (0,1), p_incr, g_incr are learning factors. Usually p_incr, g_incr = 2. However, the values of p_incr, g_incr are problem-dependent. As it can be easily seen, higher values of g_incr help particles to move out of local minima. Thus, usually in our experiments, we have taken higher values of g_incr.

Particles' velocities on each dimension are clamped to a maximum velocity Vmax. If the sum of accelerations cause the velocity on that dimension to exceed Vmax, which is a parameter specified by the user, then the velocity on that dimension is limited to Vmax.

In our implementation, each particle corresponds to a key. The fitness of a key is found by one of the fitness functions discussed in the previous section. Since DES uses seven byte key, our search space is 7-dimensional. Thus each byte corresponds to a separate direction. Vmax is varied throughout the experiments. The range varied is from 8-64.

Remember that a particle can take a maximum of 256 different values in each direction.

3.5.2 p_incr and g_incr

As we have seen, two parameters we can play around in PSO are p_incr and g_incr. They are so called learning factors of the algorithm. Higher values of p_incr allow a particle to move towards *its* direction of search faster. Higher values of g_incr allow all the particles to move in the direction of group leader faster. These are very essential parameters in PSO. In a discrete optimisation problem such as cryptanalysis of DES, it is better to have higher values of g_incr.

3.5.3 The Group PSO

Here we discuss a unique variation of Particle Swarm Optimisation. In the PSO algorithm that we have seen, there is a single swarm made up of large number of particles. Experiments show that increasing the number of particles in a single swarm does not give better results. It only adds to more computation. In order to divide the search space, we explore the option of "group of swarms", a concept proposed by us for the first time.

In this concept, there are many swarms active at a given time. The swarms are initialised in such a way that they are evenly distributed across the search space. In ideal case, their movements should not overlap. To achieve this, proper distribution of swarms across the search space is important. Each of the swarms moves through the search space independently irrespective of other swarms. Each swarm has its own global best and rest of the particles in that swarm try to follow that global best.

The algorithm provided remains the same except that there are many versions of this algorithm running at the same time. This is an ideal situation for multiprocessor systems or grids. However, in our experiments we have implemented the group swarm algorithm for sequential execution. The speedup achieved by the parallel version or on a grid could be an interesting thing to investigate. Since there is very little communication among swarms, there are many opportunities to modify the algorithm so that it suits well for parallel or distributed execution.

4 EXPERIMENTS AND RESULTS FOR SDES

Following are the results obtained using different heuristic techniques. The success of retrieving the key is discussed under each case below. The timings reported are on a timeshared configuration; thus relative performance can be considered.

4.1 Tabu Search

Tabu search performed consistently better than other heuristics. The average time taken by tabu search to find the correct key was about 27 seconds. It took on an average 344 decryptions for finding the correct key. It is significantly better than 512 (average) decryptions in brute force attack. Note that, in this case, brute force attack is very simple and quite efficient too. Tabu search was consistent in the sense that it always provided the answer within the range of 20-40 seconds.

4.2 Particle Swarm Optimisation

The second heuristic tried was PSO. PSO performed well in some cases and performed badly in others. PSO has given answers in 5 seconds in the best case, whereas at times it took as long as 50 seconds to get the correct keys. The performance of PSO depends upon the random seed that has to be given to the algorithm. Unfortunately, there are no guidelines about how to give this seed. PSO on an average took 400 generations performing better than the brute force method.

4.3 Adaptive Genetic Algorithm

Adaptive Genetic Algorithm with periodic changes in pc and pm values extracted the key in 42 seconds on an average. It required 460 decryptions to get the correct key. AGA, thus, was worse than both PSO and tabu search but it was slightly better than brute force attack in terms of number of function evaluations (decryptions).

4.4 Genetic Algorithm

Genetic Algorithm did not perform better than brute force method. Timings are not mentioned here because parameters given to GA were such that number of function evaluations should be less than 1024 – the worst case for brute force. However, GA was unable to find the solution in 1024 function evaluations.

5 EXPERIMENTS AND RESULTS FOR MODIFIED DES

5.1 Adaptive Genetic Algorithm

Experiments with same parameters as GA were performed. We provide some representative results here. AGA with fitness function-based changes of parameters was run for cryptanalysis of 6 round DES. In this particular experiment, population size was 500.There are considerable improvements over normal GA. Fitness value started at about 0.85 and it came down to almost 0.4.

Also, it can be seen that the number of times the change that occurred in best fitness value of a generation is also quite high. This asserts the fact that AGA has come out of local minima many times.

In another experiment performed, we used a different fitness function named number of intelligible characters. That resulted in lower values of fitness function. In the experiment, fitness value started from around 0.63 and has come down to 0.3 in 500 generations. AGA with periodic changes in pc and pm has performed a lot better.

5.2 Tabu Search

We start with a random key which is also added to the tabu list and name it as current key. We find out the fitness function of this key. Fitness function used may be any of those discussed in section 3.1.In the original tabu algorithm, there are no specifications regarding how to perturb a particular solution. In this implementation, we randomly perturb random number of bits in the current key to form a set of candidate keys. For this, a random number, n is generated. This number indicates the number of candidate solutions that are to be generated. To generate each of these candidate solutions, we complement one bit of the current key. The bit to be complemented is also randomly selected.

Once this set of candidate keys is formed, we calculate the fitness of each candidate key. We select the best key out of this set of candidate keys. We check whether this best key is present in the tabu list. If not, this best key is added to the tabu list and this best key becomes our current key. We perform the same steps as above with this key.

If the best key is in the tabu list and aspiration criterion is not satisfied (we shortly describe the aspiration criterion), then we look for the next best key in the set of candidate keys. This procedure is continued till a suitable candidate for the next current key is found or set of candidate keys is exhausted. If the set of candidate keys is exhausted, we do not change the current key and perform different random perturbation with the same key.

We keep track of the best key obtained so far, so that we can return this key at the end of the algorithm. The algorithm may be terminated by a limit on the number of iterations or if the correct key is found i.e. if fitness of a key is less than some very low threshold (about 0.08)

Tabu list size is the only one parameter we can manipulate in tabu search. The larger size of tabu list causes a lot of memory overhead. A very high value may even lead to segmentation faults. With smaller size of tabu list, the list gets full quickly; hence we cannot continue the run for too long. If the old keys are deleted to make room for new keys, we may re-search the same search space destroying the whole purpose of tabu list. Thus, tabu list size is one critical parameter to choose.

5.2.1 Aspiration Criterion

Aspiration criterion used in tabu search is always problem-specific. In this case, we say that aspiration criterion is satisfied if we do not get a better solution than the current best solution for 5 iterations. This number 5 is just an example. Experiments have been performed with different limits on aspiration criterion to be satisfied. Thus, if aspiration criterion is satisfied, we allow the best candidate key to be the current key even if that candidate key is in the tabu list.

The concept of aspiration criterion is important in tabu search to avoid getting stuck in local minima. The conjecture is that the aspiration criterion gives some indication about whether the algorithm is trapped in local minima or not. Careful thought must be given while selecting the aspiration criterion. In our case, if limit on the aspiration criterion is too small, then the algorithm may unnecessarily change a correct direction. Large limit on aspiration criterion makes the algorithm to stay in local minima for longer than necessary duration. In the search space of DES keys, there are large number of local minima. Also, global minimum is due to a discrete jump. Thus, a smaller than usual limit on aspiration criterion is advisable.

Similar sets of experiments were performed with tabu search as with GA and AGA. In this case, tabu list size was fixed at 1000 and the limit on aspiration criterion was 5, i.e. if we did not get a better solution in five consecutive generations, aspiration criterion is said to be satisfied. The fitness function used in this case is Unigram.

5.3 PSO

PSO and group PSO were used for cryptanalysis of 6-round DES. We summarize the results of these

experiments in this subsection. The fitness function used was unigram. The fitness value started from about 0.5 and come down to 0.3 in about 3250 generations. Also, we notice a rapid decrease in the fitness function at the beginning and the rate of decrease decreases as we run the experiments for longer period. This is because as the fitness value decreases, it becomes more and more difficult to find a better key. This is the problem with all the algorithms. PSO, in spite of being so simple, has performed better than other methods.

In case of PSO, search space was divided into 100 swarms. At the beginning, it was ensured that swarms start at different positions. But it is very difficult to ensure that trajectories of the swarms do not intersect during the execution. The fitness value has started from above 0.55 and has come down to 0.22. Number of function evaluations is also quite high in this case. Thus, PSO is a very promising approach for solving the problems of cryptanalysis of DES and any discrete optimisation problem in general. However, this is a fairly new technique and there is lot of scope for research in this area. One of the ideas suggested in this paper, is the exploration of group PSO in a parallel execution environment.

The group PSO as compared to GA and AGA performs better in terms of percentage of successful attacks, overall execution time, and need to tune various parameters of the algorithm leading to simple program development effort.

6 CONCLUSIONS

Though the keys of 16-bit DES were retrieved in all experiments using different heuristics, tabu search and particle swarm optimisation performed better than the other methods in terms of execution time.

The paper has demonstrated the efficacy of evolutionary computation principles in cryptanalysis studies. We have considered DES and one of its simplified variants, along with SDES for our experiments. Initially it was noticed that cryptanalysis of DES was hard using genetic algorithm, adaptive genetic algorithm, particle swarm optimisation, and tabu search technique. However, the attack was successful for SDES and simplified version of DES with 6 rounds,16 bit key and without S-boxes. For a comparison of the results obtained from the evolutionary techniques, we have considered results from Simulated Annealing technique as the benchmark. We notice that evolutionary techniques yield better results. These comparison details are not presented here due to space constraints. This demonstrates that it is possible to cryptanalyse block ciphers using evolutionary techniques. Currently we are experimenting with 32 bit and 48 bit key DES with more rounds and S-boxes. We believe that such studies will certainly establish the applicability of evolutionary computation techniques to cryptanalysis studies.

REFERENCES

[1] Clark A and Dawson Ed, “Optimisation Heuristics for the Automated Cryptanalysis of Classical Ciphers”, Journal of Combinatorial Mathematics and Combinatorial Computing, Vol. 28,pp. 63-86, 1998.

[2] Glover Fred, Taillard Eric and Werra Dominique de, ”A User’s Guide to Tabu Search” Annals of Operations Research, Vol. 41,pp. 3-28,1993.

[3] Goldberg D.E, “Genetic Algorithms in Search, Optimisation and Machine Learning”, Boston, Addison-Wesly, 1989.

[4] James Kennedy and Russell Eberhart, Particle Swarm Optimisation, Proceedings of the IEEE International Conference on Neural Networks, 1995, pp.1942-1948.

[5] Kirkpatrick S, C .D. Gelatt. Jr. and Vecchi M. P, ”Optimisation by Simulated Annealing”, Science, Vol. 220, No. 4598, pp. 671-680,1983.

[6] Spillman R,”Cryptanalysis of Knapsack Ciphers using Genetic Algorithms”, Cryptologia, Vol.17, No.4, pp. 367-377, 1993.

[7] Spillman R, Janssen M, Nelson B and Kepner M, “Use of Genetic Algorithm in the Cryptanalysis of Simple Substitution Ciphers”, Cryptologia, Vol. 17, No.1, pp. 30-44.1993.

[8] Srinivas M and L. M. Patnaik, “Adaptive probabilities of crossover and mutation in genetic algorithms,” *IEEE Trans. Syst., Man Cybern.*, vol. 24, pp. 656-667, Apr. 1994.

[9] Srinivas M and Patnaik L.M,”Genetic Algorithms: A Survey”, IEEE Computer, pp.17-26, 1994.

[10] William Stallings, Cryptography and Network Security Principles and Practices, Third Edition, Pearson Education Inc.,2003.