A Modified Chaotic Firefly Algorithm for Solving Discrete Logarithm Problem and Analysis

Mohit Mishra, Utkarsh Chaturvedi, K. K. Shukla
Dept. of Computer Science and Engineering
IIT (BHU) Varanasi
Varanasi, India
{mohit.mishra.cse11,
utkarsh.chaturvedi.cse11}iitbhu.ac.in,
kkshukla.cse@iitbhu.ac.in

R. V. Yampolskiy
Dept. of Computer Engineering and Computer Science
University of Louisville
Louisville, KY USA
roman.yampolskiy@louisville.edu

Abstract—
In this paper, we present a modified version of firefly algorithm that shows considerable potential in solving discrete logarithm problem, a mathematical function used in public-key cryptography like Diffie-Hellman Key Exchange and ElGamal Encryption. Firefly Algorithm has been experimentally proved to have outperformed a number of metaheuristics like the popular Particle Swarm Optimization. While solving the problem of finding discrete logarithm, we also evaluate the effectiveness of the algorithm and its modified version in solving such cryptographic problems. Observations show significant potential of Firefly Algorithm in solving small instances of the problem, while it calls for further research in scaling up the effectiveness of the algorithm in solving bigger instances of the problem. Simultaneously, we also analyze the convergence of the modified algorithm.

I. MOTIVATION

The discrete logarithm problem (DLP) [1] finds frequent applications in public-key cryptography [2], notably in Digital Signatures [2], ElGamal Encryption [3], and Diffie-Hellman Key Exchange [2]. The problem can be mathematical put as following. Given a multiplicative group G modulo prime p with g and y being t eh elements of the group where g is the generator of the prime p, the problem requires to find such an integer x ∈ [0, p) that satisfies the following equation:

$$y = g^x (mod\ p) \quad (1)$$

It is worth noting that the DLP is a one way mathematical function [3] similar to the factoring problem, which makes it computationally difficult to compute the answer. Essentially, a one-way mathematical function is easy to compute in one direction, but computationally difficult to do so in the reverse direction. The problem of discrete exponentiation is an easy task and can be computed efficiently using exponentiation by squaring for instance. However, computing the inverse problem of discrete logarithm is computationally hard. Modern cryptography relies on such mathematical functions.

Quite notably, there have been exact algorithms in literature devised for solving DLP, namely Baby-Giant Step Algorithm

[4], Pollard's Rho Algorithm [5], Pollard's Lambda Algorithm [6], Pohlig-Hellman Algorithm [7], Index Calculus Algorithm [8], Number Field Sieve [9], and Function Field Sieve [10]. A brief survey of the current state of art can be found in [11]. The Number Field Sieve is the fastest known algorithm for computing discrete logarithms on prime fields. Its asymptotic time complexity is given by $L_p\left[\frac{1}{3}, \left(\frac{64}{9}\right)\left(\frac{1}{3}\right)\right]$, where

$$L_p[v, \delta] = exp\left(\left(\delta + o(1)\right)(\log p)^v (\log\log p)^{1-v}\right) \quad (2)$$

Further, Cheon [12] showed that, for g being an element of prime order p in an abelian group and $x \in Z_p$, if g, g^x, and g^{x^d} are given for a positive divisor d of $p-1$, we can compute the secret x in $O\left(\log p \,.\left(\frac{p}{d} + \sqrt{d}\right)\right)$ group operations using $O\left(max\left\{\frac{p}{d}, \sqrt{d}\right\}\right)$ memory. If g^{x^i} $(i = 0, 1, 2, \ldots, d)$ are provided for a positive divisor d of $p+1$, x can be computed in in $O\left(\log p \,.\left(\frac{p}{d} + \sqrt{d}\right)\right)$ group operations using $O\left(max\left\{\frac{p}{d}, \sqrt{d}\right\}\right)$ memory.

Consider Shank's algorithm, if p = 170141183460469231731687303715884105727, it would take 1.14824E21 steps for it to solve for $d = 1$. The memory complexity of Shank's algorithm is $O\left(\sqrt{p}\right)$.

Such exact algorithms although guarantee the optimality of solutions obtained, incur huge cost upon memory and runtime for computationally hard problems. To deal with this, there exist a class of approximation algorithms like metaheuristics, which although *don't* guarantee the optimality of solutions obtained, provide reasonably approximate and acceptable solutions in reasonable (strictly speaking, polynomial) time.

Metaheuristics serve five main purposes [13]:

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.262

1. Solving problems quickly,

2. Solving large complex problems, and

3. Obtaining robust algorithms.

While the first two points are about "problems", obtaining robust algorithms depend on the sensitivity of the algorithm against input instances and its parameters. In general, robustness of an algorithm can be defined as how sensitive the algorithm is against small deviations in the input data or parameters of the algorithm. In the metaheuristic community, robustness also measures the performance of the algorithms given different types of inputs. In case of stochastic algorithms, robustness can be measured as the average deviation of the behavior of the algorithm over different runs of the algorithm on the "same" instance. For more understanding, one may refer to [13]. Metaheuristics, in general, are easy to design and implement, and at the same time they are flexible in nature, which is why it makes them so popular in their usage.

Some of the most notable swarm intelligence metaheuristics are Ant Colony Optimization (ACO) [14], Artificial Bee Colony (ABC) [15], Particle Swarm Optimization (PSO) [16], and Firefly Algorithm [17]. These algorithms have been successfully implemented and tested on many NP-Hard and NP-Complete problems like Traveling Salesman Problem, Knapsack Problem, Set Covering, Clique Problem, Hamiltonian-Cycle Problem, Vehicle Routing, Subset Problem, etc.

Speaking of cryptographic problems, the problem with the design of objective functions for problems of cryptography is that the nature of the function is binary – correct/incorrect [18], Hence solving problems like prime factorization and DLP is difficult owing to the difficulty in the optimal design of objective functions. In literature, there exists some contributions towards solving these problems using metaheuristics as discussed later.

There has been a growing interest among the researchers in employing metaheuristics for solving hard mathematical functions like the prime factorization problem and the DLP problem. As far as DLP is concerned, the first and only such contribution is one due to Laskari [19] where the author made use of Differential Evolution and PSO. There has been a growing interest in the research community towards exploring and exploiting the potential metaheuristics have in solving problems of cryptography like prime factorization [26], [27], [28], [29], [30] and [18]. A constructive critic study on such algorithms being applied to such problems can be found in [31].

This paper introduces a modified version of the Firefly Algorithm to adapt it for usage of such problems like DLP itself, and further evaluate the algorithm's effectiveness both in terms of performance and convergence. Summing up, our contributions are the following:

- Establishing the superiority of firefly algorithm over PSO.
- Using Firefly Algorithm for solving discrete logarithm problem, which is represented as an integer programming problem.
- Significant improvement of performance the modified version of the algorithm as compared to the classical firefly algorithm when used for problems like DLP.
- We also establish the convergence property of the algorithm, which to the best of our knowledge, is the first such contribution in general, to the theory of convergence of firefly algorithm in terms of theory and not just empirical analysis.

This paper is divided into the following sections: Section II presents the design of the solution proposed in a high level view. Section III gives a brief introduction to the Firefly Algorithm and our modified chaotic version of the algorithm. Section IV describes with the formulation of the objective function for DLP. Section V provides the test cases which we tested upon. Section VI presents experiments and observations. In Section VII, we provide a convergence analysis of the algorithm thus applied to DLP. We finally conclude about our results and future work in Section VIII.

II. DESIGN

In this paper, we present DLP as an instance of integer programming problem (IPP) [20] which essentially converts the problem to a discrete optimization problem. The DLP can be formulated as a single variable integer programming problem. Let us define $h(x)$ as:

$$h(x) = y - g^x \tag{3}$$

We model our objective function around h(x) which is discussed later in Section III. Fig. 1 presents the function plot for $p = 101, g = 3, y = 17, f(x) = h(x) mod(p)$.

We use a modified chaotic firefly algorithm (M-CFA) for DLP with some modifications introduced into the classical firefly algorithm:

1. We always move a firefly stochastically towards the global best firefly in any iteration.

2. When a firefly overshoots the domain space during its movement, we distribute the firefly in the domain space randomly instead of clamping the firefly to the boundary value of the search domain.

3. Introduce chaotic factor (using logistic maps) in the parameters of the algorithm.

Preliminary experimentation suggested superiority of chaotic firefly algorithm over other swarm based metaheuristics like PSO, hence we have concentrated our work on further analysis and refinement of the algorithm to suit problems like DLP.

III. Firefly Algorithm

The Firefly Algorithm [17] is a recent addition to the family of swarm-based metaheuristics. It is inspired by the flashing behavior of the fireflies. The less bright fireflies get attracted towards brighter fireflies taking into account the media around the problem domain. Three assumptions are made while designing the algorithm:

1. All fireflies are unisex, such that each firefly gets attracted to all other fireflies.
2. The attractiveness factor is proportional to the brightness of the firefly. A less bright firefly will get attracted to a brighter firefly. Also, as the distance between two fireflies increase, the attractiveness factor

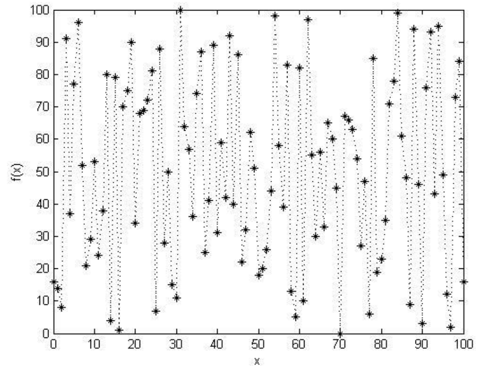

Fig. 1: Function plot of $f(x) = (3^x - 17)mod(101)$

between them reduces since the intensity of brightness reduces, which is inversely proportional to the square of the distance between the fireflies.
3. Landscape of the problem domain also affects the movement of the fireflies. That is, a media with a large absorption coefficient will reduce the intensity of brightness of the fireflies, thereby hampering movement.

The firefly algorithm has been exploited to solve a number of hard optimization problems. Suppose there are N fireflies, that is, N candidate solutions. Each candidate solution xi has a cost associated with it which is obtained from the objective function of the problem under consideration. The objective function determines the brightness function I_i of each firefly:

$$I = f(x_i)$$

Based on the computed fitness values, the less bright firefly gets attracted and moves towards the brighter one. At the source, the brightness is higher than any other point in the environment. This brightness decreases with increase in the distance between any two fireflies. At the same time, surrounding media absorbs some amount of light determined by an absorption coefficient, γ. For two fireflies i and j, the fitness values vary inversely with distance between the two fireflies r_{ij}. Since light intensity follows the inverse square law, we have the light intensity as a function of the distance between any two fireflies, *r*, as:

$$I(r) = \frac{I_0}{r^2} \tag{4}$$

where I_0 is the light intensity at the source. By incorporating the media absorption coefficient γ, light intensity can be represented in the Gaussian Form as:

$$I(r) = I_0 e^{-\gamma r^2} \tag{5}$$

The attractiveness function thus becomes:

$$\beta(r) = \beta_0 e^{-\gamma r^2} \tag{6}$$

where β_0 is the attractiveness value at $r = 0$.

In our case we always move the fireflies towards the brightest firefly, and the dimension of the problem is just one since DLP is a single variable problem. Hence the movement equation is formulated as:

$$x_{i+1} = \lfloor x_i + \beta(x_{best} - x_i) + \alpha S(rand_i - 0.5) \rfloor \tag{7}$$

where $rand_i$ is a random number in [0,1) generated for the i^{th} firefly. The term with α is called the randomization parameter. *S* is the scaling parameter equal to the difference between the upper and lower bounds on *x*. The solution is floored to obtain an integral solution. The distance between two fireflies is calculated by their Cartesian distance.

Also, in our approach, we randomize the solution vectors which happen to cross the boundary limits of the domain space during any iteration, instead of freezing them to the boundary values.

Following is the algorithm code for the modified version of the firefly algorithm:

```
Initialize the algorithm parameters α, β and γ.
while (t < Maximum_Generation)
        for i = 1 : n
            if  (f($x_i$) < f($x_{best}$))
              Compute the attractiveness values from Eq. (7)
      Move firefly i towards the current best fire fly in
              accordance with Eq.(7)
        If the firefly moves out of the boundary of domain
              space during any iteration, distribute the
              firefly randomly in the domain space.
              Evaluate the new fitness values and update
              the solutions
   end if
```

```
        end for
    Update α and γ in accordance with Eq. (9) and Eq.(10).
        Rank the according to their fitness values and find the
        current best firefly
end while
```

IV. Adaptation of M-CFA for DLP

A. Objective Function

We define our objective function as:

$$f(x) = \begin{cases} h(x) mod\, p & if\ h(x) \geq 0 \\ p - h(x) mod\, p & if\ h(x) < 0 \end{cases} \quad (8)$$

where *h(x)* is the function defined earlier in Eq. (3), and we are given the values of p, y and g. g is the generator of the prime p. We need to figure out the value of x in the interval [0, p-2] for which the Eq. (1) satisfies. Now with this objective function, we have DLP as an integer programming problem to optimize for.

B. Integer Programming

The movement equation of the firefly algorithm consists of an attraction term and a randomization term. The equation produces a real value which is floored to obtain an integral solution as described in Eq. (7).

C. Parameters and Chaos

We introduce chaos [21] in the parameters α and γ using Logistic Maps [22, 23], and update them after each iteration as follows:

$$\alpha_{t+1} = \mu_1 \alpha_t (1 - \alpha_t) \quad (9)$$

$$\gamma_{t+1} = \mu_2 \gamma_t (1 - \gamma_t) \quad (10)$$

where t is the sample time, μ_1 and μ_2 are the control parameters [0, 4].

The incorporation of chaos in the parameters has been well studied in [24]. Basically, it helps the algorithm to escape any local minima if it gets trapped in. There are other Chaotic Maps as well which can be incorporated for the same problem. Logistic Maps are simple to implement and provide similar results as other maps do, which were verified in the course of experimentation.

V. Test Cases

Following table presents the test cases we used for our experiments. These test cases are derived from [19]. The test cases were verified before commencing our experiments. The experiments performed are preliminary in nature since the objective was to understand the potential of the algorithm and further tuning it, followed by convergence analysis.

TABLE I: TEST CASES FOR EXPERIMENTS

Test Case	p	g	y	x
T1	1009	11	337	784
T2	2003	5	558	1552
T3	3001	14	1001	2032
T4	4001	3	1334	3999
T5	9973	11	3324	5658
T6	10007	5	3336	3816
T7	99991	6	33330	72140
T8	500029	6	166676	365276
T9	1000003	2	333334	245724

For each test case, we evaluate $\sqrt{p}\log p$ [5] to represent the number of steps needed to solve DLP theoretically, for example, by Shank's Algorithm. We shall compare this with our approach using M-CFA.

TABLE II: NUMBER OF STEPS TAKEN BY SHANK'S ALGORITHM

Test Case	p	$\sqrt{p}\log p$
T1	1009	316.97
T2	2003	490.87
T3	3001	632.79
T4	4001	756.90
T5	9973	1326.59
T6	10007	1329.34
T7	99991	5252.15
T8	500029	13387.09
T9	1000003	19931.60

VI. Observations

We experimented our method on Java 1.7.0_25; Java HotSpot™ Client VM 23.25-b01 on NetBeans IDE 7.3.1 on a 64 bit OS, x64-based processor, Windows 8 version 6.2, Intel® Core™ i5-3210 CPU @ 2.50 GHz 2.50 GHz, 4.00 GB (3.88 GB usable) RAM with no networks attached.

We set the M-CFA parameters as α = 0.6, γ = 0.01, β_0 = 1.0, and μ = 4.0.We compare the performance of the algorithm with the global variant of Particle Swarm Optimization. We set the algorithm parameters as w (inertia weight) = 0.2, c_1 (cognitive factor) = c_2 (social factor) = 2. We also experimented with other values, but we achieved best results with the parameters used in the experiments.

For each dataset, 50 trials are taken and hit ratio is calculated accordingly with each hit corresponding to a success in computing x. The mean iterations is calculated on the basis of the iterations obtained only in the success scenarios.

The observations of our experiment are shown below in the observation table. SS represents the swarm size for the firefly algorithm and PSO, MAX-ITER represents the maximum number of iterations for the given test case and MI represents the mean number of iterations computed. Note that the mean iterations are calculated on the basis of the iterations obtained only in the successful scenarios.

TABLE III: COMPARISON BETWEEN PSO AND M-CFA PERFORMANCE: HIT RATIO (HR) AND MEAN ITERATIONS (MI) FOR TEST CASES T1-T9.

Test Case	SS	MAX-ITER.	PSO		M-CFA	
			HR	MI	HR	MI
T1	30	500	41/50	204.98	50/50	44.96
T2	45	500	43/50	147.23	50/50	54.96
T3	55	500	34/50	170.94	50/50	55.10
T4	70	700	31/50	119.68	50/50	128.12
T5	100	1000	32/50	90.62	50/50	102.26
T6	100	1000	26/50	57.19	50/50	97.82
T7	350	2000	16/50	286.81	50/50	225.98
T8	750	3500	14/50	1006.93	50/50	424.62
T9	1000	5000	2/50	174.00	50/50	905.96

We observe that M-CFA beats the global variant of PSO with a much greater efficiency factor, which suggests the superiority of M-CFA over PSO. Interestingly, the algorithm's hit ratio reduces as soon as our domain goes beyond 10^8. Following is the new case we tested (T10) the algorithm upon:

p = 10486717, g = 7, y= 546423, for which x = 645613. MAX-ITER = 7500

With increasing number of fireflies, the hit ratio increases considerably. We achieved a 48/50 hit ratio when the firefly population is increased to 5000 in T10 wit mean iterations of 2286.85 as compared to the 35/50 hit ratio with 2695.25 mean iterations when the number of fireflies being 2000, and 25/50 hit ratio with 2677.92 mean iterations and 1200 fireflies for the same test case T10. This shows that given sufficient resources, the algorithm has potential to solve the DLP problem.

Below is the graph that shows the variation in the number of iterations and number of fireflies for p = 1000003, g = 2 and y = 333334. The mean iterations is calculated for 10 successful trials for a given firefly population without setting any condition for maximum allowable iterations. The firefly population is arithmetically increased by 10 for each set of 10 trials. The graph clearly depicts a trade-off between the mean iterations and the number of fireflies, which suggests that an optimal choice of maximum iterations and firefly population is essential for efficiency of the algorithm adapted for DLP.

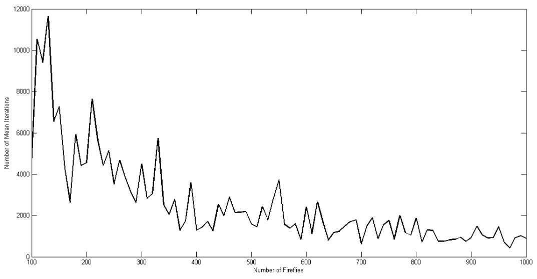

Fig. 2: Variation in the number of iterations and number of fireflies for p = 1000003, g = 2 and y = 333334

The modified version of the FA where we move each firefly only towards the global best firefly also has a significant advantage over the classical firefly as shown below:

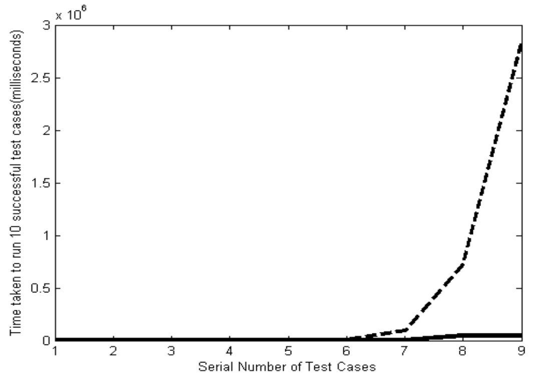

Fig. 3: Time Complexity Analysis of Classical Chaotic FA versus M-CFA

The solid line depicts the Modified Chaotic Firefly Algorithm (M-CFA) we have adapted for DLP and the dotted line represents the classical firefly algorithm with the same parameters and chaos. The graph is computed based on the 9 test cases mentioned in the observation table. The time is calculated (in milliseconds) over 10 trials with successful hits without the limit on the maximum number of iterations. The graph clearly depicts that the modified version of the firefly algorithm possess the property of scalability and is almost constant in nature with a very slight increase in slope after 7th test case, whereas the classical algorithm shows an exponential increase in the time to reach to the solution. Both the approaches gave similar hit ratios but different scalability. This presents a crisp idea that the modified version of our chaotic firefly algorithm possesses huge promising potential in solving DLP owing to its scalability. The classical firefly algorithm's performance closely represents the Taylor's expansion of exponential around 8 (retaining terms upto 7th degree) where x represents the input size coupled with a factor of firefly population:

$$time = \frac{x^7}{5040} + \frac{x^6}{720} + \frac{x^5}{120} + \frac{x^4}{24} + \frac{x^3}{6} + \frac{x^2}{2} + x + 1 \qquad (11)$$

The graph of the above estimation is shown below:

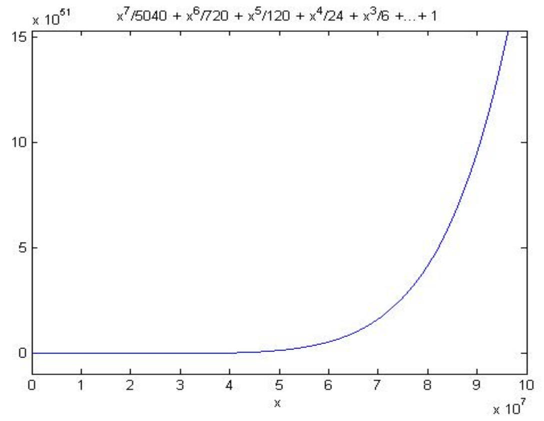

Fig. 4: Taylor's expansion of exponential around 8

So, we can predict the order of time that the classical firefly algorithm will take for a given firefly population to converge to the solution as in Eq. (10), in contrast to almost constant (with slight increase in slope) order of time of M-CFA. Based on the time computed, the observation table below presents the improvement percentage for each test case over 10 trials of successful hits:

TABLE IV: COMAPRISON BETWEEN CLASSICAL FIREFLY ALGORITHM AND M-CFA FOR DLP BASED ON TIME ANALYSIS AND IMPROVEMENT PERCENTAGE

Test Case	Mean Time for Classical FA with Chaos (in ms)	Mean Time for M-CFA (in ms)	Improvement Percentage
T1	17.9	5.6	219.64%
T2	34.4	9.5	262.10%
T3	162.5	13.4	1112.69%
T4	551.6	37.5	1371.00%
T5	626.6	23.4	2577.78%
T6	6610.0	52.2	1166.28%
T7	9256.8	549.4	1584.90%
T8	72613.4	4466.7	1525.66%
T9	282564.2	4739.7	5861.65%

VII. CONVERGENCE ANALYSIS

To understand the convergence of the algorithm, we make use of the theory behind discrete dynamical systems [25]. Dynamical systems are about the evolution of quantities under consideration over time, which can occur over time or in discrete time steps. In case of discrete time steps, we describe the system as discrete dynamical system.

To model the evolution, we take snapshots of the system at a sequence of discrete time stamps, thereby recording the state of the system. The state variables evolve through the state space.

The movement equation of the fireflies can be transformed in the form of a vector matrix as follows:

$$\begin{bmatrix} x^{t+1} \\ x^t \\ 1 \end{bmatrix} = \begin{bmatrix} 1-\beta & 0 & \beta x_{best} + \alpha\, S\,(rand_i - 0.5) \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^t \\ x^{t-1} \\ 1 \end{bmatrix} \quad (12)$$

Let A the matrix as:

$$A = \begin{bmatrix} 1-\beta & 0 & \beta x_{best} + \alpha\, S\,(rand_i - 0.5) \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (13)$$

According to Cayley-Hamilton's Theorem, every matrix satisfies its own characteristic equation. So the characteristic equation of the matrix A is:

$$|\lambda I - A| = 0 \quad (14)$$

where λ is the set of eigenvalues that satisfies the characteristic equation, and I is the identity matrix.

So the characteristic equation for our recurrence relation becomes:

$$\begin{vmatrix} \lambda+\beta-1 & 0 & \beta x_{best} + \alpha\, S\,(rand_i - 0.5) \\ 1 & \lambda & 0 \\ 0 & 0 & \lambda-1 \end{vmatrix} = 0 \quad (15)$$

$$\lambda(\lambda-1)(\lambda+\beta-1) = 0$$
(16)

Hence, we get $\lambda = \{0,\ 1,\ 1-\beta\}$. 0 and 1 are trivial solutions. The other solution is $\lambda = 1-\beta$, which is less than 1. Since every eigenvalue is between 0 and 1, hence we have a stable discrete dynamical system.

Now for any iterative method to be convergent, the spectral radius ρ should lie in the interval (0,1). Hence, we have:

$$|1-\beta| < 1 \quad (17)$$

or,

$$0 < \beta < 2 \quad (18)$$

So the condition for the convergence of our algorithm is that β should lie in the interval (0, 2).

The explicit recurrence relation of Eq. (7) can therefore be given as:

$$x(t+1) = k_1 + k_2(1-\beta)^t \quad (19)$$

where k_1 and k_2 are constants determined from the initial conditions of the problem in concern.

Now since β (0,2), therefore $|1-\beta|$ is always less than 1. Now for t → ∞, x converges to k_1. Hence, the algorithm converges to the required solution, and therefore the algorithm is convergent in nature when $0<\beta<2$.

We verified our results on the same data set. We set the maximum iterations to 10,000 in each case. In each test case, it failed to converge to a solution.

One may question that showing convergence in infinite time steps does not bear much practical importance, however, the basic idea behind showing the convergence of the algorithm is that the algorithm is "convergent" in nature. Analytically, the algorithm showed to hit a success in all the test cases described earlier.

A further insight into the stability of the system can be achieved by talking about the stationary state. The stationary state for a general discrete system $x_{t+1}=f(x_t)$ is a $\bar{x}=f(\bar{x})$. So, if evaluate the stationary state for our movement equation, it comes out to be as:

$$\bar{x}=x_{best}+\frac{randomization\ term}{\beta} \tag{20}$$

Hence we do see here the stationary point is close to the best value, which is obvious. However, it also depends on the randomization and attractiveness factor. This suggests that a fine tuning of the algorithm can lead to effectiveness and efficiency. Any deviation from the parameters can lead to divergence. Parameter setting hence become a critical part of the algorithm.

VIII. Conclusion

The observations clearly show the potential that the Firefly Algorithm possesses in solving the Discrete Logarithm Problem. The success ratios of each data set is impressive which depicts that fine tuning and parameter selection can lead to more accuracy and scalability. The convergence analysis show that the algorithm is convergent in nature for 0<β<2. Given sufficient resources to the Firefly Algorithm, the algorithm possesses a significant potential in cracking through the DLP. The most significant advantage of adapting metaheuristics like Firefly Algorithm is that such algorithms can be easily parallelized and therefore can be used to test on bigger numbers. Further investigation needs to be done on bigger dataset.

A significant advantage that algorithms like Firefly Algorithm is that they can be easily parallelized and thereby can reduce the computational time of execution. It is worth noting that how firefly algorithm is able to solve the problem (though on small instances as tested) even though the nature of problem is binary in nature. The question then becomes: What characteristic of the objective function makes it so suitable for the firefly algorithm, and vice versa. Often, defining and clear understanding of the behavior of such algorithms are difficult to evaluate theoretically, though empirical analysis shows a significant potential of the algorithm.

As a part of our future work, we intend to pursue a comprehensive investigation about how metaheuristics in general perform in finite field, rings and finite groups, including groups derived from elliptic curve cryptography. While approximation algorithms provide an "approximate" yet practically acceptable solutions, there is no such concept of "near" or "approximate" solution in a finite field. Hence, it then becomes imperative to study the behavioral nature of the metaheuristics on theoretical grounds rather than mere empirical grounds. This, as a result, would elicit opinions on how metaheuristics can be transformed or modified to be applied on problems of cryptography in which the algebraic structures belong to finite fields, rings and groups.

On the other hand it also becomes equally important as to how one can design an effective objective function for such binary problems to which the answer is just a "yes" or "no". As discussed in Section II, it makes sense to transform a binary problem like DLP to an optimization problem, whereon metaheuristics can come into picture to solve the problem. It is, however, because of the nature of the problem and/or the inability to evolve a "good" fitness function that these problems of cryptography are difficult to solve without using sufficient memory and runtime resources.

References

[1] McCurley, K. S. "The Discrete Logarithm Problem", Proceedings of Symposia in Applied Mathematics, vol. 42, pp. 49-74, 1990.

[2] Diffie, W., Hellman, W., "New Directions in Cryptography", IEEE Transactions on Information Theory, vol. 22 (6): 644–654, 1976.

[3] ElGamal, T, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms", *IEEE Transactions on Information Theory* **31** (4): 469–472, 1985.

[4] S. Goldwasser and M. Bellare, Lecture Notes on Cryptography, July 2008, retrieved from http://cseweb.ucsd.edu/~mihir/papers/gb.pdf in Jan-Feb 2013.

[5] D. Shanks, Class number, a theory of factorization and genera. In Proc. Symp. Pure Math. 20, 1969, pages 415–440. AMS, Providence, R.I., 1971.

[6] J. M. Pollard,. "Monte Carlo methods for index computation (mod *p*)". Mathematics of Computation 32 (143): 918–924, 1978

[7] Mollin, Richard (2006-09-18). "An Introduction To Cryptography" (2nd ed.). Chapman and Hall/CRC. p. 344

[8] L. Adleman, "A subexponential algorithm for the discrete logarithm problem with applications to cryptography", In 20th Annual Symposium on Foundations of Computer Science, 1979

[9] Arjen K. Lenstra and H. W. Lenstra, Jr. (eds.). "The development of the number field sieve". Lecture Notes in Math. (1993) 1554. Springer-Verlag.

[10] L. Adleman, M. A. Huang, "Function Field Sieve Method for Discrete Logarithms over Finite Fields", Information and Computation, vol. 151, issues 1−2, pp. 5-16, 25 May 1999.

[11] A. Odlyzko, "Discrete Logarithms: The past and the future", Design. Codes, and Cryptography, vol. 19, issue 2-3, pp. 129-145.

[12] Cheon, J. H., "Security Analysis of the Strong Diffie-Hellman Problem", EUROCRYPT 2006, LNCS 4004, pp. 1-11.

[13] Talbi, E., “Metaheuristics From Design to Implementation”, John Wiley & Sons, 2009.

[14] Dorigo, M., Birattari, M., Stutzle, T., “Ant Colony Optimization,” IEEE Computational Intelligence Magazine, vol. 1 issue 4, pp. 28-39, Nov. 2006.

[15] Karoboga, D., Basturk, B., “A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm”, Journal of Global Optimization, Springer US, vol. 39, pp. 459-471, Nov. 2007.

[16] J. Kennedy, R. Eberhart, “Particle Swarm Optimization”, IEEE International Conference on Neural Networks, Perth WA, 1995, vol. 4, pp. 1942-1948.

[17] X. S. Yang, “Firefly algorithms for multimodal optimization”, Stochastic Algorithms: Foundations and Applications, SAGA 2009, Lecture Notes in Computer Sciences, vol. 5792, pp. 169-178, 2009.

[18] R. V. Yampolskiy, “Application of bio-inspired algoritm to the problem of intger factorization”, International Journal of Bio-inspired Computation. Vol. 2, No. 2, pp. 115-123, 2010

[19] E.C. Laskari et al, “The Discrete Logarithm Problem as an Optimization Task: A First Study”, Proceedings of the IASTED International Conference on Artificial Intelligence and Applications (AIA 2004), Innsbruck, Austria, ACTA Press (IASTED 2004)

[20] Garfinkel, R. S., Nemhauser, G. L., “Integer Programming”, John Wiley & Sons, New York, 1972.

[21] L. dos S. Coelho, D. L. de A. Bernert, V. C. Mariani, “A Chaotic Firefly Algorithm Apllied to Reliability Redundancy Optimization”, 2011 IEEE Congress on Evolutionary Computation (CEC), New Orleans LA, pp. 517-521, June 2011.

[22] T. S. Parker and L. O. Chua, “Practical numerical algorithms for chaotic system,” Springer-Verlag, Berlin, Germany, 1989.

[23] S. H. Strogatz, Nonlinear dynamics and chaos, Perseus Publishing, Massachussetts, 2000.

[24] Fister Jr, I., Perc, M., Kamal, S. M., Fister, I., “A review of chaos-based firefly algorithms: Persepectives and research challenges”, Applied Mathematics and Computation, Elsevier, vol252, pp. 155-165, 2015.

[25] Galor, O., “Discrete Dynamical Systems”, Springer, 2007.

[26] Chan, D. M., “Automatic generation of prime factorization algorithms using genetic pro-gramming,” Genetic Algorithms and Genetic Programming at Stanford. pp.52–57, Stanford Bookstore. Stanford, California, 2002.

[27] Meletiou, G., Tasoulis, D. K., Vrahatis, M. N, “A first study of the neural network approach to the RSA cryptosystem,” IASTED 2002 Conference on Artificial Intelligence, Banff, Can-ada, 2002, pp.483–488.

[28] Jansen, B, Nakayama, K., “Neural networks following a binary approach applied to the integer prime-factorization problem,” IEEE International Joint Conference on Neural Networks (IJCNN), July 2005, pp.2577–2582.

[29] Laskari, E. C., Meletiou, G. C., Tasoulis, D. K., Vrahatis, M. N., “Studying the performance of artificial neural networks on problems related to cryptography,” Nonlinear Analysis: Real World Applications, Vol. 7, pp.937–942, 2006.

[30] Mishra, M, Chaturvedi, U., Pal, S. K., “A Multithreaded Bound Varying Chaotic Firefly Algorithm for Prime Factorization,” IEEE International Advance Computing Confer-ence (IACC), Feb 2014, Gurgaon, pp. 1321-1324.

[31] Mishra, M., Chaturvedi, U., Gupta, V., Shukla, K. K., Yampolskiy, R. V., “A Study on the Limitations of Evolutionary Computation and Other Bio-inspired Approaches for Integer Factoriztion“, In the proceedings of 2015 International Conference on Soft Computing and Software Engineering, March 5-6, 2015, Berkeley, CA USA. In press.