The Discrete Logarithm Problem as an Optimization Task: A First Study

E.C. Laskari[(1,3)], G.C. Meletiou[(2,3)], M.N. Vrahatis[(1,3)]
[(1)] Department of Mathematics, University of Patras, GR–26110 Patras, Greece,
[(2)] A.T.E.I. of Epirus, P.O. Box 110, GR–47100 Arta, Greece,
[(3)] University of Patras Artificial Intelligence Research Center (UPAIRC),
University of Patras, GR–26110 Patras, Greece
Email: elena@math.upatras.gr, gmelet@teiep.gr, vrahatis@math.upatras.gr

Abstract

Most of the contemporary cryptographic systems are based on mathematical problems whose solutions are generally intractable in polynomial time; such problems are the discrete logarithm problem and the integer factorization problem. In this contribution we consider the discrete logarithm problem as an Integer Programming Problem. Two Evolutionary Computation methods, namely the Particle Swarm Optimization and the Differential Evolution algorithm, as well as the Random Search technique, are employed as a first approach to tackle this new Integer Programming Problem. Results indicate that this new approach is promising.

1 Introduction

Cryptography is indispensable in numerous developing fields, such as electronic transactions, e-commerce, e-business, e-voting and others, and has thus become a key technology for the emerging information technologies society.

A number of hard and complex computational problems have been motivated by public key cryptography. The assumption that these problems are in general computationally intractable in polynomial time forms the basis of the reliability of contemporary cryptosystems. Such problems are the integer factorization problem related to the RSA cryptosystem; the index computation or the discrete logarithm problem related to the El Gamal cryptosystem, as well as, to the Diffie–Hellman key exchange; knapsack problems, and others [2, 4, 19, 20]. These problems originate from different fields of mathematics, including computational algebra, computational number theory, probability, mathematical logic, Diophantine complexity, algebraic geometry, etc.

The algebraic structures which are used for the representation of cryptosystems are finite fields, $\mathbb{Z}_p$ or $GF(p,k)$, rings $\mathbb{Z}_N$ and finite groups, including groups derived from elliptic curves. In general, computation on such structures is very hard. For example, in a finite field we cannot find a non-trivial topology, which renders it a topological field (that is making addition and multiplication continuous functions). The same holds for a non–trivial order or a non–trivial distance. Concepts like *"near"*, *"greater than"*, *"less than"*, *"approximation"*, *"convergence"* have no meaning in a finite field. This is one of the reasons why finite fields are applied in cryptography. Traditional computational tools are designed for the approximation of real (complex) numbers, or functions, and therefore cannot be directly applied to computations over finite fields.

The discrete logarithm problem (DLP) is defined as follows: given a prime p, a generator α of $\mathbb{Z}_p^*$, and an element $\beta \in \mathbb{Z}_p^*$, find the integer x, $0 \leqslant x \leqslant p-2$, such that

$$\alpha^x \equiv \beta \pmod{p}.$$

The first efforts for the solution of the DLP led to the development of algorithms like Shanks's baby–step giant–step algorithm, Pollard's rho algorithm and their variants [15, 16, 22], which were later classified as generic. These algorithms require at most $O(p^{1/2})$ group operations and they are optimal since $O(p^{1/2})$ has been proved to be a lower bound for generic algorithms [24]. Non–generic algorithms, namely the index–calculus algorithms, have also been developed. These algorithms achieve a subexponential running time and the Number Field Sieve version [7] is the asymptotically fastest known method for the computation of the discrete logarithms in prime fields. Its running time is given by $L_p[1/3, (64/9)(1/3)]$, where

$$L_p[\nu, \delta] = \exp((\delta + o(1))(\log p)^{\nu}(\log\log p)^{1-\nu}).$$

A survey of the current state of the art in discrete logarithms can be found in [11, 12].

In this contribution we consider the DLP as an Integer Programming Problem (IPP) and a first approach to tackle it through Evolutionary Computation methods and Random Search. Evolutionary Computation algorithms are stochastic optimization methods that involve algorithmic mechanisms inspired by natural evolution and social behavior respectively. They can handle problems that involve discontinuous objective functions and disjoint search spaces [5, 9, 21]. Commonly encountered paradigms of such methods are Genetic Algorithms (GA), Evolution Strategies (ES), Differential Evolution algorithm (DE) and the Particle Swarm Optimization (PSO). GA and ES are based on the principles of natural evolution. On the other hand, PSO is based on the simulation of social behavior. Optimization techniques for real search spaces can be applied on Integer Programming problems through slight modification. Usually, the optimum solution is determined by rounding off the real optimum values to the nearest integer [18]. Early approaches in the direction of Evolutionary Algorithms for Integer Programming problems are reported in [6, 8].

In this paper we study the performance of two Evolutionary Computation methods, namely PSO and DE, as well as the Random Search technique to solve the DLP. The rest of the paper is organized as follows. In Section 2 the PSO and DE algorithms are briefly described. In Section 3 the formulation of the DLP as an optimization problem is given and experimental results are reported. In Section 4 conclusions and insights for further work are derived.

2 The Evolutionary Computation methods considered

For completeness purposes, in this section we briefly describe the two Evolutionary Computation methods.

PSO is a population–based algorithm that exploits a population of individuals, to search promising regions of the function space. In this context, the population is called *swarm* and the individuals are called *particles*. Each particle moves with an adaptable velocity within the search space, and retains in its memory the best position it ever encountered. In the *global* variant of the PSO the best position ever attained by all individuals of the swarm is communicated to all the particles. In the *local* variant, each particle is assigned to a neighborhood consisting of a prespecified number of particles. In this case, the best position ever attained by the particles that comprise the neighborhood is communicated among them [3].

Assume a D–dimensional search space, $\mathbf{S} \subset \mathbb{R}^D$, and a swarm of N particles. The i–th particle is in effect a D–dimensional vector $X_i = (x_{i1}, x_{i2}, \dots, x_{iD})^\top$. The velocity of this particle is also a D–dimensional vector, $V_i = (v_{i1}, v_{i2}, \dots, v_{iD})^\top$. The best previous position ever encountered by the i–th particle is a point in $\mathbf{S}$, denoted by $P_i = (p_{i1}, p_{i2}, \dots, p_{iD})^\top$. Assume g, to be the index of the particle that attained the best previous position among all the individuals of the swarm. Then, according to the *constriction factor* version of PSO the swarm is manipulated using the following equations [1]:

$$\begin{aligned} V_i^{(t+1)} &= \chi \left(V_i^{(t)} + c_1 r_1 \left(P_i^{(t)} - X_i^{(t)} \right) + \right. \\ &\quad \left. + c_2 r_2 \left(P_g^{(t)} - X_i^{(t)} \right) \right), \qquad (1) \\ X_i^{(t+1)} &= X_i^{(t)} + V_i^{(t+1)}, \qquad (2) \end{aligned}$$

where $i = 1, 2, \dots, N$; χ is the constriction factor; c_1 and c_2 denote the *cognitive* and *social* parameters respectively; r_1, r_2 are random numbers uniformly distributed in the range $[0, 1]$; and t, stands for the counter of iterations.

The value of the constriction factor is typically obtained through the formula $\chi = 2\kappa/|2 - \phi - \sqrt{\phi^2 - 4\phi}|$, for $\phi > 4$, where $\phi = c_1 + c_2$, and $\kappa = 1$. Different configurations of χ as well as a theoretical analysis of the derivation of the above formula, can be found in [1].

In a different version of PSO, a parameter called *inertia weight*, is used, and the swarm is manipulated according to the formulae [3, 9, 23]:

$$\begin{aligned} V_i^{(t+1)} &= wV_i^{(t)} + c_1 r_1 \left(P_i^{(t)} - X_i^{(t)}\right) + \\ &\quad + c_2 r_2 \left(P_g^{(t)} - X_i^{(t)}\right), \qquad (3) \\ X_i^{(t+1)} &= X_i^{(t)} + V_i^{(t+1)}, \qquad (4) \end{aligned}$$

where $i = 1, 2, \ldots, N$; and w is the inertia weight, while all other variables are the same as in the constriction factor version. There is no explicit formula for the determination of the factor w, which controls the impact of the previous history of velocities on the current one. However, since a large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, (fine–tuning the current search area) it appears intuitively appealing to initially set it to a large value and to gradually decrease it to obtain more refined solutions. The superiority of this approach against the selection of a constant inertia weight, has been experimentally verified [23]. Thus, an initial value around 1.2 and a gradual decline toward 0.1 can be considered as a good choice for w.

Proper fine–tuning of the parameters c_1 and c_2, results in faster convergence and alleviation of local minima. As default values, $c_1 = c_2 = 2$ have been proposed, but experimental results indicate that alternative configurations, depending on the problem at hand, can produce superior performance [9, 13].

Typically, the swarm and the velocities, are initialized randomly in the search space. For uniform random initialization in a multidimensional search space, a Sobol Sequence Generator can be used [17]. Recently, the performance of the PSO method for the IPP was studied in [10] with very promising results.

The DE algorithm has been developed by Storn and Price [25]. It is a parallel direct numerical search method, which utilizes N, D–dimensional parameter vectors $x_{i,G}$, $i = 1, \ldots, N$, as a population for each iteration (generation) of the algorithm. The initial population is taken to be uniformly distributed in the search space. At each generation, the *mutation* and *crossover* (*recombination*) operators are applied on the individuals, giving rise to a new population, which is subsequently subjected to the selection phase.

According to the *mutation* operator, for each vector $x_{i,G}$, $i = 1, \ldots, N$, a *mutant vector* is generated through the equation:

$$v_{i,G+1} = x_{r_1,G} + F\,(x_{r_2,G} - x_{r_3,G}), \qquad (5)$$

where $r_1, r_2, r_3 \in \{1, \ldots, N\}$, are mutually different random indexes, and, $F \in (0, 2]$. The indexes r_1, r_2, r_3, also need to differ from the current index, i. Consequently, to apply mutation, N must be greater than, or equal to, 4.

Following the mutation phase, the crossover operator is applied on the population. Thus, a *trial vector*,

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Di,G+1}), \qquad (6)$$

is generated, where,

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, \text{if } (\mathrm{randb}(j) \leqslant CR) \text{ or } j = \mathrm{rnbr}(i), \\ x_{ji,G}, \quad \text{if } (\mathrm{randb}(j) > CR) \text{ and } j \neq \mathrm{rnbr}(i), \end{cases} \qquad (7)$$

where, $j = 1, 2, \ldots, D$; $\mathrm{randb}(j)$, is the j–th evaluation of a uniform random number generator in the range $[0, 1]$; *CR* is the (user specified) crossover constant in the range $[0, 1]$; and, $\mathrm{rnbr}(i)$ is a randomly chosen index from the set $\{1, 2, \ldots, D\}$.

To decide whether or not the vector $u_{i,G+1}$ will be a member of the population of the next generation, it is compared to the initial vector $x_{i,G}$. Thus,

$$x_{i,G+1} = \begin{cases} u_{i,G+1}, & \text{if } f(u_{i,G+1}) < f(x_{i,G}), \\ x_{i,G}, & \text{otherwise.} \end{cases}$$

The procedure described above is considered as the standard variant of the DE algorithm. Different mutation and crossover operators have been applied with promising results [25]. In order to classify the different variants, the scheme *DE*$/x/y/z$ is used, where x specifies the mutated vector ("rand" for randomly selected individual or "best" for selection of the best individual); y is the number of difference vectors used; and, z denotes the crossover scheme (the scheme described here is due to independent binomial experiments, and thus, it is denoted as "bin") [25]. According to this description scheme, the DE

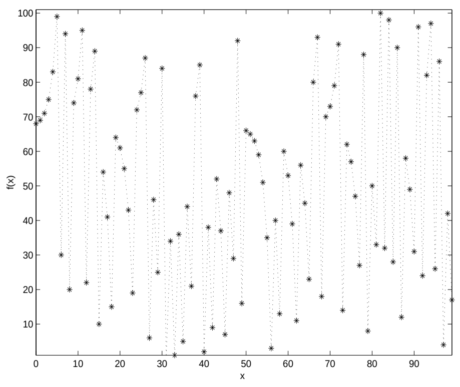

Figure 1: Plot of the function $f(x) = \alpha^x - \beta \pmod p$, for $p = 101,\ \alpha = 2$ and $\beta = 34$.

variant described above is denoted as $DE/rand/1/bin$. One highly beneficial scheme that deserves special attention is the $DE/best/2/bin$ scheme, where

$$v_{i,G+1} = x_{best,G} + F\,(x_{r_1,G} + x_{r_2,G} - x_{r_3,G} - x_{r_4,G}).$$

The usage of two difference vectors seems to improve the diversity of the population, if N is large enough.

3 Problem formulation and experimental results

The discrete logarithm problem (DLP) can be formulated as the minimization of a function

$$f : \mathbb{Z}_p^* \mapsto \{0, 1, \ldots, p-1\},$$

where

$$f(x) = \alpha^x - \beta \pmod p,$$

p is a given prime, α is the generator of $\mathbb{Z}_p^*$ and $\beta \in \mathbb{Z}_p^*$ also given. The global minimum of the function f is zero and the corresponding global minimizer is the discrete logarithm of β to base α modulo p. Thus, the DLP is transformed to an IPP on a bounded set. The global minimizer of f is unique since α is a generator. Notice that the value of f for a given integer x can be easily obtained by the repeated square–and–multiply algorithm for exponentiations in $\mathbb{Z}_n$ [11]. Although, the DLP can be considered as an one-dimensional minimization problem, it is not an easy task. The function f produced by the DLP exhibits almost pseudo random behavior. For example, an illustration of the function f for $p = 101,\ \alpha = 2$ and $\beta = 34$ is given in Fig. 1. Furthermore, the safety of the cryptosystems based on the DLP requires large encryption keys. The size of the encryption keys is proportional to the number of digits of the prime number p of the DLP. Thus, in the minimization of f we are mostly interested in large primes p which make the optimization procedure quite difficult. An optimization algorithm capable of finding the global minimizer of f for large primes p with the smallest cost, suffices to break all the cryptosystems based on the corresponding discrete logarithms.

The two Evolutionary Computation methods considered, were applied on the DLP by rounding off real values to the nearest integer. The global and local PSO variants of both the inertia weight and the constriction factor versions, as well as the $DE/rand/1/bin$ and $DE/best/2/bin$ variants of the DE algorithm, have been used. Random Search technique has also been tested. For the velocity of the PSO method a lower bound $V_{min} = 1$ has been set to prevent premature convergence to local minima. All populations were constrained in the feasible region of the problem.

The performance of the methods was investigated for prime numbers p, in $p = 1009$ to $p = 1000003$. For each prime number p considered, 100 independent runs were performed and the corresponding results are exhibited in Tables 1,2. Concerning the notation used in the Tables; PSOGW corresponds to the global variant of PSO method with inertia weight; PSOGC is the global variant of PSO with constriction factor; PSOLW is PSO's local variant with inertia weight; PSOLC is PSO's local variant with constriction factor, DE1 corresponds to the $DE/rand/1/bin$ and DE2 to the $DE/best/2/bin$ variants of DE method. Random Search results are denoted as RS. A run is considered to be successful if the algorithm has identified the global minimizer within a prespecified number of function evaluations. Two function evaluation thresholds are imposed, $\log_2^2 p$ and $\log_2^3 p$. The success rates of each algorithm considered, that is the proportion of the times it achieved the global minimizer within the prespecified thresholds, are reported in the Tables. From the percentage $c\%$ of success rates reported, it is clear that the DLP can be broken if $100/c$ times the corresponding threshold function evaluations are performed. In Fig. 2

the total computational cost required to break the DLP for various primes is exhibited in logarithmic scale.

		Success rate for	
Problem	Method	$\leqslant \log_2^2 p$	$\leqslant \log_2^3 p$
	PSOGW	7%	61%
	PSOGC	6%	63%
$p = 1009$	PSOLW	3%	55%
$\alpha = 11$	PSOLC	7%	60%
$\beta = 337$	DE1	4%	37%
	DE2	1%	45%
	RS	15%	68%
	PSOGW	4%	60%
	PSOGC	7%	68%
$p = 2003$	PSOLW	5%	63%
$\alpha = 5$	PSOLC	6%	63%
$\beta = 668$	DE1	5%	23%
	DE2	1%	24%
	RS	9%	46%
	PSOGW	3%	20%
	PSOGC	2%	21%
$p = 3001$	PSOLW	3%	17%
$\alpha = 14$	PSOLC	2%	18%
$\beta = 1001$	DE1	1%	15%
	DE2	1%	8%
	RS	3%	36%
	PSOGW	3%	28%
	PSOGC	3%	14%
$p = 4001$	PSOLW	2%	27%
$\alpha = 3$	PSOLC	3%	15%
$\beta = 1334$	DE1	1%	8%
	DE2	1%	10%
	RS	2%	33%
	PSOGW	1%	20%
	PSOGC	2%	35%
$p = 9973$	PSOLW	2%	15%
$\alpha = 11$	PSOLC	3%	28%
$\beta = 3324$	DE1	1%	11%
	DE2	0%	5%
	RS	5%	21%

Table 1: Success rates of each method for two different thresholds of function evaluations for $p = 1009$ to $p = 9973$.

4 Conclusions

Most of the contemporary cryptographic systems are based on mathematical problems whose solutions are generally intractable in polynomial time. The discrete logarithm problem is one of these problems and is related to the El Gamal cryptosystem and to the Diffie–Hellman key exchange. In this paper the discrete logarithm problem is formulated as an Integer Programming Problem which is further addressed as a minimization task. For this purpose two Evolutionary Computation methods, the Particle Swarm Optimization and the Differential Evolution algorithm, as well as Random Search technique, were used. Evolutionary Computation techniques have the advantage

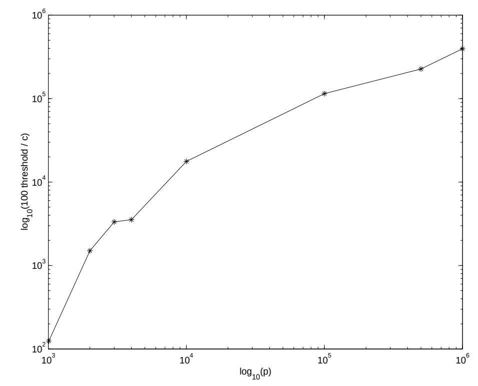

Figure 2: Computational cost required to break the DLP for various primes, in logarithmic scale.

		Success rate for	
Problem	Method	$\leqslant \log_2^2 p$	$\leqslant \log_2^3 p$
	PSOGW	1%	23%
	PSOGC	1%	20%
$p = 10007$	PSOLW	1%	24%
$\alpha = 5$	PSOLC	1%	20%
$\beta = 3336$	DE1	1%	10%
	DE2	0%	6%
	RS	2%	27%
	PSOGW	0%	3%
	PSOGC	1%	7%
$p = 99991$	PSOLW	0%	3%
$\alpha = 6$	PSOLC	1%	5%
$\beta = 33330$	DE1	0%	4%
	DE2	0%	2%
	RS	1%	3%
	PSOGW	0%	2%
	PSOGC	0%	2%
$p = 500029$	PSOLW	0%	2%
$\alpha = 6$	PSOLC	0%	3%
$\beta = 166676$	DE1	0%	2%
	DE2	0%	0%
	RS	0%	2%
	PSOGW	0%	1%
	PSOGC	0%	2%
$p = 1000003$	PSOLW	0%	1%
$\alpha = 2$	PSOLC	0%	2%
$\beta = 333334$	DE1	0%	0%
	DE2	0%	0%
	RS	0%	0%

Table 2: Success rates of each method for two different thresholds of function evaluations for $p = 10007$ to $p = 1000003$.

that they do not require gradient information and can operate on discontinuous and disjoint search spaces [13]. The performance of these methods was tested on several instances of the problem and the obtained results indicate that this problem can be confronted in time no more than subexponential. A significant advantage of these methods is that they can be readily parallelized thereby reducing significantly the time required for their execution [14]. This paper presents the first results of an ongoing research effort. Numerous issues remain unresolved. Most importantly the performance of the algorithms on very large prime numbers need to be investigated.

References

[1] M. Clerc and J. Kennedy, *The particle swarm–explosion, stability, and convergence in a multidimensional complex space*, IEEE Trans. Evol. Comput. **6** (2002), no. 1, 58–73.

[2] W. Diffie and M.E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **IT-22** (1976), no. 6, 644–654.

[3] R.C. Eberhart, P. Simpson, and R. Dobbins, *Computational intelligence pc tools*, Academic Press, 1996.

[4] T. ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31** (1985), no. 4, 469–472.

[5] D.B. Fogel, *Evolutionary computation: Towards a new philosophy of machine intelligence*, IEEE Press, Piscataway, NJ, 1995.

[6] D.A. Gall, *A practical multifactor optimization criterion*, Recent Advances in Optimization Techniques (T.P. Vogl, ed.), 1966, pp. 369–386.

[7] D. Gordon, *Discrete logarithms in* $\mathrm{GF}(p)$ *using the number field sieve*, SIAM J. Discrete Math. **6** (1993), 124–138.

[8] R.C. Kelahan and J.L. Gaddy, *Application of the adaptive random search to discrete and mixed integer optimization*, International Journal for Numerical Methods in Enginnering **12** (1978), 289–298.

[9] J. Kennedy and R.C. Eberhart, *Swarm intelligence*, Morgan Kaufmann Publishers, 2001.

[10] E.C. Laskari, K.E. Parsopoulos, and M.N. Vrahatis, *Particle swarm optimization for integer programming*, Proceedings of the IEEE 2002 Congress on Evolutionary Computation (Hawaii, HI), IEEE Press, 2002, pp. 1576–1581.

[11] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of applied cryptography*, CRC Press series on discrete mathematics and its applications, CRC Press, 1996.

[12] A. Odlyzko, *Discrete logarithms: The past and the future*, Designs, Codes, and Cryptography **19** (2000), no. 2–3, 129–145.

[13] K.E. Parsopoulos and M.N. Vrahatis, *Recent approaches to global optimization problems through particle swarm optimization*, Natural Computing **1** (2002), 235–306.

[14] V.P. Plagianakos and M.N. Vrahatis, *Parallel evolutionary training algorithms for "hardware–friendly" neural networks*, Natural Computing **1** (2002), 307–322.

[15] J.M. Pollard, *Monte carlo methods for index computation* $\pmod p$, Mathematics of Computation **32** (1978), no. 143, 918–924.

[16] ———, *Kangaroos, monopoly and discrete logarithms*, Journal of Cryptology **13** (2000), 437–447.

[17] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical recipes in fortran 77*, Cambridge University Press, 1992.

[18] S.S. Rao, *Engineering optimization–theory and practice*, Wiley Eastern, New Delhi, 1996.

[19] R.L. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, Communications of the ACM **21** (1978), 120–126.

[20] B. Schneier, *Applied cryptography*, second ed., John Wiley and Sons, Inc., New York, 1996.

[21] H.-P. Schwefel, *Evolution and optimum seeking*, Wiley, New York, 1995.

[22] D. Shanks, *Class number, a theory of factorization, and genera*, In Proceedings of Symposium in Pure Mathematics, vol. 20, American Mathematical Society, 1971, pp. 415–440.

[23] Y. Shi and R.C. Eberhart, *A modified particle swarm optimizer*, Proc. IEEE Conference on Evolutionary Computation (Anchorage, AK), IEEE Service Center, 1998.

[24] V. Shoup, *Lower bounds for discrete logarithms and related problems*, Lecture Notes in Computer Science **1233** (1997), 256–266.

[25] R. Storn and K. Price, *Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces*, J. Global Optimization **11** (1997), 341–359.