ISSN 2007-9737

Minimum Addition Chains Generation Using Evolutionary Strategies

Mauricio Olguín-Carbajal[1], Juan Carlos Herrera-Lozada[1], Israel Rivera-Zárate[1], J. Felix Serrano-Talamantes[1], Rodrigo Cadena-Martínez[2], J. Irving Vásquez-Gómez[3,4]

[1] Instituto Politécnico Nacional,
Centro de Innovación y Desarrollo Tecnológico en Cómputo,
Mexico

[2] Universidad Tecnológica de México, Mexico

[3] Instituto Politécnico Nacional, Mexico

[4] Consejo Nacional de Ciencia y Tecnología, Mexico

{molguinc, jlozada, irivera}@ipn.mx, jfserranotal@gmail.com

Abstract. The calculus for a power of a number could be a time and computational cost-consuming task. A method for reducing this issue is welcome in all mayor computational areas as cryptography, numerical series and elliptic curves calculus, just to mention a few. This paper details the development of a minimum length addition chains generator based on an Evolutionary Strategy, which makes fewer calls to the objective function with respect to other proposals that also use bio-inspirated algorithms as Particle Swarm Optimization or a Genetic Algorithm. By using fewer calls to the objective function, the number of calculations is lower and consequently decreases the generation time providing an improvement in computational cost but obtaining competitive results.

Keywords. Minimum length, addition chains, evolutionary strategy, computational cost reduction.

1 Introduction

In modern cryptography calculating numbers with exponents is widely used as part of the steps of encryption and decryption, as in the case of asymmetric cryptography algorithms such as RSA, El Gamal or DSA. In the case of RSA, keys of 512, 1024, 2048 or 4096 bits in length are used, although it is currently recommended to use 2048 or 4096 bits. If RSA is used and a key length of 1024 bits is chosen, two random numbers of 512 bits must be chosen which will be named **p** and **q**. Multiplying these numbers yields **a** = **p.q**.

The encryption key is also a randomly chosen number such that (**p**-1) (**q**-1) and **e** are relative prime numbers. In this way the decryption key **d** will be obtained from **e**, **p** and **q** as follows:

$$\boldsymbol{d} = \boldsymbol{e}^{-1} \bmod ((\boldsymbol{p}-1)(\boldsymbol{q}-1)). \quad (1)$$

The **d** and **e** numbers are used to build up the private key. To encrypt a message **M**, it must be divided into smaller parts than **a**; each block of cipher text will be obtained by:

$$C_i = M_i^{e} \bmod \boldsymbol{a}. \quad (2)$$

To decrypt it must be used each block of text coded C_i with:

$$M_i = C_i^{d} \bmod \boldsymbol{a}. \quad (3)$$

Efficient and fast approximations are needed for the calculation of large exponent numbers. If, for example, we want to obtain an exponent of $\mathbf{x}^{n}$, the simplest method is to use **n**-1 multiplications of **x** such that we have $\mathbf{x}^{n}$ **= x * x * x * x * ... * x**.

If it is considered that for a symmetric key encryption-decryption process one can have a 1024-bit **e** or **d** for each data block, then a number of 1.79×10^{308} is available.

Considering the above, we can have up to 308 consecutive multiplications per block of data to be encrypted. While some approaches focus on the creation of hardware that solves multiplications more efficiently, other approaches are focused on reducing the number of them. A method that reduces the number of operations necessary for the calculation of numbers with large powers is welcome in this field, but not only here, it is also applicable to the elliptic curves [1] as well as to the calculation of the numerical successions of Lucas and Fibonacci [2] and many other groups that use element calculations in cyclic groups. Some approximations for the reduction of multiplications in order to obtain $\mathbf{x}^n$ are: the binary method [3], the m-ary method [3] or the window-based method [4].

Although these methods significantly reduce the number of multiplications required to obtain a power, it is possible to further reduce the number of such multiplications by using exponential methods based on series of additions, in which positive integers starting at 1 and end in the exponent **n** to generate it as the last member of the series:

$$\beta = \alpha^e \bmod \boldsymbol{n}. \quad (4)$$

The set of multiplications thus obtained is called the Addition Chain, where **e** is the length of the chain, meaning **e** as the number of multiplications necessary to obtain **x** and **y** defining the length of the chain as $l(\boldsymbol{e})$. Therefore, a small number will reflect a smaller number of multiplications necessary to obtain **x** and **y** as **e** is reduced, in the same proportion the number of multiplications necessary to obtain this power is reduced.

The paper is organized as follows: section II is formally defined the problem. Section III evolutionary strategies and their implementation are described for generating addition chains; The experiments and results are presented in Section IV, finally in section V we included the conclusions and future works.

1.1 Approach to the Problem

Cruz-Cortés et al. [4] indicate that "formally an addition chain **e** of length **l** is a sequence **u** of positive integers" such that:

$u_0 = 1, u_1 ..., u_l = e$

Such that for each $i > 1$,

$u_i = u_j + u_k,$

for some **j** and **k** that satisfy $0 \leq \boldsymbol{j} \leq \boldsymbol{k} < i$.

Considering the above, if $\boldsymbol{u}$ is an addition chain, which calculates **e**, then for each $\alpha \in [1, \boldsymbol{n}-1]$ it is possible to find $\boldsymbol{\beta = \alpha^e} \bmod \boldsymbol{n}$ by successively calculating $\alpha, \alpha^{u1}, ..., \alpha^{u1-1}, \alpha^e$.

If $\boldsymbol{l(e)}$ is defined as the smallest valid length for an addition chain for a positive integer **e**, then the theoretical minimum number of multiplications required to compute the modular exponentiation, is precisely $\boldsymbol{l\,(e)}$, see equation 4. The problem is to construct an addition chain for $\boldsymbol{l\,(e)}$ of minimum length given an integer **e**. If there is more than one sequence with the same length, then either is acceptable. For example, <1,2,3,5> and <1,2,4,5>, both are valid solutions when the addition chain for the integer 5 is requested.

The optimal algorithm for the calculation of addition chains requires less multiplication than a binary power calculation for large exponents. The first example where this is evident is in the calculation of x^{15}, where the binary method requires 6 multiplications but an optimal addition chain needs only 5 multiplications [4]. However, the calculation of an addition chain is more complex. There is not currently known method for finding arbitrary exponents, and the problem of finding an optimal sequence for an addition chain has been shown to be a complete NP problem [5].

The problem of generating an optimal addition chain has not been solved, with dynamic programming since it is not enough to decompose the power into smaller powers, each optimally calculated. Since the addition chains for smaller powers may be related each. However a proposed solution to this problem is the use of bio-inspired programming. This article presents the calculation of Addition Chains using Evolutionary Strategies (ES) due to the fast convergence of these, which results in competitive results with a lower cost of computation time compared to other bio-inspired heuristics.

Computational algorithms have already been developed to find short addition chains in both the deterministic and stochastic parts. For the deterministic part we can highlight: method of factor [6], binary method [7], window method and

sliding window [8]. Among the stochastic or non-deterministic methods can be mentioned: Genetic Algorithm (GA) [4, 9], Particle Swarm Optimization (PSO) [10], Ant Colony Optimization (AC) [11], Artificial Immune System (AIS) [12] and Genetic Programming (GP) [13]. The aforementioned methods have found smaller addiction chains than deterministic methods such as the factor method or the binary method; however, most of the above work is focused on the processing of numbers with small exponents. When using an ES it is proposed to improve the time performance of a stochastic method with respect to the aforementioned ones, which allows obtaining chains of short addition in a time smaller than the one used by other stochastic bio-inspired methods.

2 Materials and Methods

ES were originally developed by a group of students at the Technical University of Berlin Germany starting in the 1970s, particularly by I. Rechenberg and H. P. Schwefel [14, 15]. Currently there are a lot of variants for the original ES [16, 17, 18]; Beyer and Schwefel make an in-depth review of the most relevant [19]. The basic algorithm of ES consists of 5 steps, see Algorithm 1 [14]:

1. Generation of the initial population.
2. Marriage.
3. Recombination.
4. Mutation.
5. Selection.

The following are the decisions made for the design of the algorithm of an ES for the calculation of addition chains.

a) ***Representation:*** In order to encode the data of each individual, whole numbers were used in the variables **x** of the ES in such a way as to have a phenotypic and genotypic representation of the problem. With this, each variable **x** represents a possible value for the calculation of the addition chain, so that each variable **x** is a possible solution to an addition chain for the next value, much like encodings in an integer array for a GA (Genetic Algorithm). For Sigma, equal values are initially used (in a range of 0 to 1.77754).

b) ***Initial population:*** The variables initial values are set by assigning a value of one to the first element and two to the second element and generating viable (valid) elements for the algorithm by selecting a random value between the previous values and double the previous value for the next element. This allows to generating valid (but not optimal) addition chains for all the individuals of the initial population.

c) ***Objective Function:*** The objective value (the power to be calculated) to be generated is the initial goal of the calculation. The objective function has to determine this value; it must also meet the conditions necessary for the chain generated to be viable:

   1. $u_0 = 1$
   2. $u_1 = \boldsymbol{e}$
   3. $u_0 < u_1 < u_2 < ... < u_{l-1} < u_l$
   4. For each k ($1 < k < m$) there exist two integers i y j (not necessarily different) with intervals $0 < i, j < k - 1$ with $u_k = u_i + u_j$

Conditions 1 to 4 define the addition chain; these conditions are coded in C language, within a loop that reviews all the genome variables of all individuals, such that they are two nested cycles, one to review each individual in each generation and the revision of each individual genome within.

d) ***Crossover operator:*** The used crossover operator consists in the selection of two parents (P1 and P2) randomly chosen from the population. These two parents are used to generate two children (H1 and H2). Generated children will compete directly between them to find the best children. A random number is chosen for parent P1 and then another random number for parent P2; being careful that it is not the same random number for both parents. The crossover operator is different from that established for canonical ES because of the unique characteristics of the problem. The main difference is that when an element is copied, the sequence of the addition chain is broken. So for the crossover it was chosen to

use the Algorithm proposed by Cruz-Cortés [4], in which the elements of an individual are not taken for the creation of the child. Instead they take the rules of the creation of the father for the termination of the child from a point of Crossing. The crossover point is selected randomly. In the next step the first chromosome of Parent 1 is copied from the beginning to the crossing point. Subsequently, the rules (not the information) of the addition chain of the parent P2 are copied from a position after the crossing point to the total length of the parent P2. The components **a** and **b** of the k^{th} element of the addition chain of the parent P2 are obtained and used to generate the element **k** of the addition chain of the child H1, always verifying that the value of the generated number does not exceed the Objective number of the addition chain. This validation is performed in order to generate viable addition chains. Finally, the same steps are taken for the creation of the H2 child but exchanging the places of the parents.

e) ***Mutation operator:*** A canonical mutation model is used based on the change in the sigma value of each chromosome per individual and then uses the updated sigma to calculate the new value of X for each gene of the individual. Mutation is performed on all variables (genome) of all individuals in a population. The mutation in the ES has two parts: the calculation of the sigma variable and the mutation of the corresponding X variable. The order of execution is: first the calculation of the new sigma and then the one of X, if done otherwise the mutation scheme does not work.

The updated sigma mutation (σ') is made from the previous sigma (σ):

$$\sigma' = \sigma \times e^{((\tau' \cdot N(0,1)) + (\tau \cdot Ni(0,1)))}, \quad (4)$$

where:

σ' corresponds to the updated sigma,

σ corresponds to the previous sigma.

The exponent is calculated using the product of a random value (N_i) with a range between zero and one multiplied by τ (tau) added to the product of another random number (N) by τ' (tau prime) as exponent of **e**.

Once calculated σ' we proceed to compute x' from the current x of the genome of the individual, where x represents a position of the vector containing the elements forming the addition chain. The probability of mutation of x is determined by the initial probability of mutation (MUT_PROB) and a random value between 0 and 100 (Mutate) that is calculated for each x, following equation 1. The x' value will be the element of the addition chain of the child H1 from which the chain begins to mutate. Due to the particular characteristics of the generation of the addition chains it is not possible to use a standard mutation for an ES, whereby the calculation of the modified chain is made from the mutated element and the later ones are calculated based on this element. Initially a random type variable (with a value between 0 and 7000) is calculated. Next, the type value is used to determine the type of addition chain to be proposed for the **CM** element (Mutated Chain), with three possible cases:

1. Generate **CM** with $H1_{k-1} + H1_{k-1,}$
2. Generate CM with $H1_{k-1} + H1_{k-2}$,
3. Generate **CM** with $H1_{k-1} + H1_{rand,}$

where:

CM is the current element to be calculated from the chain:

$H1_{k-1}$ is the previous element of the son's chain1,

$H1_{rand}$ is any element less than k in the chain of the child H1.

The intervals of type privilege the first case over the others and privileges the second over the third, since a random value between 0 and 7000 for type are calculated as follows:

1. if 0 < **type** < 5000 then **CM** = $H1_{k-1} + H1_{k-1}$,
2. if 5001 < type < 6500 then CM = $H1_{k-1} + H1_{k-2}$,
3. if 6501 < **type** < 7000 then **CM** = $H1_{k-1} + H1_{rand.}$

Once having the mutated element, (**CM**) of the current chain, is checked to see if the calculated value is greater than the target value, if so there is calculated again the **CM** element to be minor than or equal to the target value.

Table 1. Statistical results for the first experiment of the ES. Calculation of addition chains for the accumulated intervals

$e \in$	Best	Avg	Media	Worst	Stand. Dev.
(1-512)	4924	4925.56	4925	4928	1.1043
(1-1000)	10815	10820.00	10819.5	10827	2.7038
(1-1024)	11121	11127.36	11127.5	11135	3.4087
(1-2000)	24105	24118.93	24117.0	24128	7.5198
(1-2048)	24779	24790.3	24791.0	24805	6.7831
(1-4096)	54625	54653.13	54656.5	54666	12.6102

Table 2. Comparative for the accumulated results of the addition chains. The algorithms compared where AIS, GA, PSO and GP

$e \in$	Optimal	AIS	GA	PSO	GP	ES
(1-512)	**4924**	4924	4924	---	4924	4924
(1-1000)	**10808**	10813	10809	---	10808	10815
(1-1024)	**11115**	11120	---	11120	11115	11121
(1-2000)	**24063**	24108	24076	---	24070	24105
(1-2048)	**24731**	24778	24748	---	24737	24779
(1-4096)	**54425**	54617	54487	---	54454	54625

When the mutated element (**CM**) is verified to be minor than or equal to the target value, **CM** is integrated into the child's addition chain as the $H1_{ik}$ element. Subsequently **CM** is compared with the target value if **CM** is equal to the target value then previous sigma (σ') is updated with the new sigma value (σ''), otherwise it will continue with the previous value, and the calculation is completed for the mutation of the addition chain.

f) ***Selection by aptitude:*** The selection is through a comparison, so that the best individuals compare parents against parents and children against children and the best of each group are those who survive to the next generation. The selection method is a bubble method in which the best abilities rise in the list and of the total of individuals only half survives, in the case that the initial population is 100 individuals, after the crossing stage have 200 individuals between parents and children. Later in the selection and generation of the new population survives only half (top 100 individuals) which is the number of individuals of the initial population again.

3 Results

To test the performance of the developed algorithm, three experiments were proposed. The first experiment consists of generating the consecutive addition chains for the intervals (1-512), (1-1000), (1-1024), (1-2048), (1-4096) and verify the accumulated value of each sequence and compare it with similar algorithms in terms of total length, and the standard deviation for thirty runs in each range. Second and third experiments were composed of thirteen target numbers for which it is known that it is difficult to calculate their corresponding addition chain.

For experiments 1 and 2, a population size of one hundred individuals, one thousand generations, thirty executions for each chain, an initial sigma of 1.7754, a mutation probability of fifteen percent, and a cross-probability of eighty percent.

Table 3. Result of ES for calculation of addition chains of difficult numbers with a thousand generations

Objective number	Length r				
	Best	Avg	Media	Worst	Stand. Dev.
11,231	18	18	18.0	18	0.0
18,287	19	19	19.0	19	0.2537
34,303	20	20	20.0	20	0.0
65,131	21	21	21.0	21	0.3051
110,591	22	22	22.0	22	0.4901
196,591	23	23	23.0	23	0.1826
357,887	24	25	24.2	25	0.5040
685,951	25	25	25.0	25	0.3457
1,176,431	26	27	26.7	27	0.4138
2,211,837	27	29	27.9	28	0.7915
4,169,527	28	29	28.8	28	0.6065
7,624,319	29	31	29.9	30	0.4842
14,143,037	30	31	30.8	31	0.5833

Table 4. ES result for hard numbers addition chains calculation using five hundred generations

Objective	Length r				
	Best	Worst	Avg	Media	Stand. Dev.
11,231	18	18	18.00	18	0.0
18,287	19	20	19.16	19	0.3790
34,303	20	21	20.06	20	0.2537
65,131	21	22	21.06	21	0.2537
110,591	22	23	22.50	22.5	0.5085
196,591	23	23	23.00	23	0.0
357,887	24	25	24.60	25	0.4795
685,951	25	26	25.23	25	0.4302
1,176,431	26	28	27.03	27	0.4138
2,211,837	27	29	28.20	28	0.7144
4,169,527	28	30	28.46	28	0.5713
7,624,319	29	32	30.50	31	0.7311
14,143,037	30	32	31.46	31	0.5561

First Experiment.

This experiment consists of generating the corresponding addition chains in the range of 1 to Maximum Range and adding the total lengths of all the obtained chains. The value obtained is the accumulated value of all the addition chains obtained for the defined interval. The ranges to be calculated are: (1-512), (1-1000), (1-1024), (1-2000), (1-2024) and (1-4096). The result is

compared with other heuristic algorithms for calculation of addition chains, see table 1 and table 2.

In the results of the experiments we present the target value, the best length reached, the worst, and the average of the 30 executions, see table 3.

Second Experiment

This experiment objective is to find the minimum addition chains for thirteen numbers, which are known to be difficult to calculate their corresponding chain. In this experiment the ES is used with a stop condition of a thousand generations to have a reference to experiment one. The objective of this second experiment is to compare the result of the ES with respect to other bioinspired algorithms in the calculation of minimum addition chains. Using difficult-to-calculate numbers that have been used elsewhere, see Table 3.

Third Experiment

This experiment uses the same ES parameters of the other two experiments. Its objective is to calculate the minimum addition chains for difficult numbers. However, the number of generations from one thousand was reduced to five hundred in order to observe the behavior of the algorithm with a stop condition of greater exigency.

The same set of thirteen hard to calculate numbers is used for which it is known that it is difficult to calculate its corresponding addition chain. The results of the experiments show the target value, the best length achieved, the worst, and the average of the 30 executions, see table 4.

4 Discussion

In the first set of experiments it can be observed that the ES algorithm obtains the addition chains for all proposed numbers, and the lengths of the chains found correspond to the best current values [20, 21, 22]. Experiments can be compared with the results obtained by other bio-inspired heuristics as reported in their respective investigations. In the case of Genetic Algorithm (GA), the data reported by Cruz-Cortés et al.

In [4] and for a Particle Swarm Optimization (PSO) algorithm, the data reported by Léon-Javier et al. in [10]. In the result of the comparison it can be seen that the minimum lengths reached by the ES are equal to those obtained with PSO and in some cases better than those obtained with an GA, see Table V.

Table 5. Comparison of three algorithms for calculating addition chains. Final length of each addition chain for each problem is shown

Objective number	Length r		
	ES	GA	PSO
11,231	18	18	18
18,287	19	19	19
34,303	20	20	20
65,131	21	21	21
110,591	22	22	22
196,591	23	23	23
357,887	24	25	24
685,951	25	25	25
1,176,431	26	27	26
2,211,837	27	28	27
4,169,527	28	29	28
7,624,319	29	30	29
14,143,037	30	31	30

Table 6. Comparison of three algorithms for calculating chains. Total calls to the target function are shown

Algorithm	Individuals/ particles	Generations	Total
PSO	30	10000	300,000
GA	100	1000	100,000
ES	100	500	50,000

When comparing other bio-inspired algorithms that calculate addition chains, it can be observed that the proposed algorithm (ES) reaches the same results, however there are some differences in the number of evaluations that are performed to obtain the addition chain.

Considering that the three algorithms for generating addition chains are very similar in their theory and implementation can be considered that the generation of a valid addition chain has the same computational cost per individual.

Table 7. Result of chains obtained for difficult exponents and their associated lengths

Exponent e = c(r)	Addition chain	Length
11231	1- 2- 3- 4- 7- 14- 28- 56- 84- 168- 252- 504- 588- 1176- 2352- 3528- 7056- 10584- 11172	18
18287	1- 2- 4- 8- 9- 18- 19- 38- 76- 152- 304- 608- 912- 1824- 3648- 7296- 10944- 18240- 18278- 18287	19
34303	1- 2- 3- 6- 8- 14- 28- 56- 112- 168- 336- 504- 1008- 2016- 4032- 8064- 16128- 32256- 34272- 34300- 34303	20
65131	1- 2- 3- 6- 8- 14- 28- 56- 112- 224- 448- 896- 1792- 3584- 3612- 7224- 14448- 21672- 43344- 65016- 65128- 65131	21
110591	1- 2- 3- 4- 8- 11- 19- 38- 76- 152- 228- 380- 760- 1140- 2280- 4560- 9120- 18240- 36480- 54720- 109440- 110580- 110591	22
196591	1- 2- 3- 6- 7- 14- 28- 56- 84- 168- 336- 504- 1008- 1512- 3024- 6048- 12096- 24192- 48384- 96768- 193536- 196560- 196588- 196591	23
357887	1- 2- 3- 5- 10- 15- 30- 45- 90- 180- 360- 720- 1440- 2880- 5760- 11520- 23040- 46080- 69120- 115200- 230400- 345600- 357120- 357840- 357885- 357887	24
685951	1- 2- 3- 5- 7- 14- 28- 42- 84- 126- 252- 504- 1008- 2016- 4032- 8064- 12096- 20160- 40320- 80640- 161280- 322560- 645120- 685440- 685944- 685951	25
1176431	1- 2- 3- 6- 9- 18- 27- 54- 108- 216- 432- 864- 865- 919- 1838- 3676- 7352- 14704- 29408- 58816- 117632- 235264- 470528- 705792- 1176320- 1176428- 1176431	26
2211837	1- 2- 4- 5- 9- 18- 36- 54- 59- 118- 127- 254- 508- 1016- 2032- 4064- 8128- 16256- 32512- 65024- 130048- 260096- 520192- 1040384- 2080768- 2210816- 2211832- 2211837	27
4169527	1- 2- 4- 6- 12- 24- 48- 96- 144- 240- 241- 385- 625- 1010- 2020- 4040- 8080- 16160- 32320- 64640- 129280- 258560- 517120- 1034240- 2068480- 4136960- 4169280- 4169521- 4169527	28
7624319	1- 2- 4- 5- 9- 18- 36- 41- 82- 164- 246- 492- 984- 1968- 3936- 7872- 15744- 15785- 31570- 47355- 94710- 189420- 378840- 568260- 947100- 1894200- 3788400- 7576800- 7624155- 7624319	29
14143037	1- 2- 3- 6- 12- 18- 30- 31- 62- 124- 248- 496- 992- 1023- 2046- 4092- 8184- 12276- 24552- 36828- 73656- 147312- 220968- 441936- 883872- 1767744- 3535488- 7070976- 14141952- 14142975- 14143037	30

If we take as a measure the number of individuals multiplied by the number of iterations / generations we have a measure of comparison. The results can be seen in Table 6 where it is observed that ES occupies a smaller number of calculations than those using an Genetic Algorithm (GA) or a Particle Swarm Optimization (PSO) algorithm, see Table VI.

Finally, it should be noted that the addition chains generated are different from those obtained by other algorithms and yet they also have minimum lengths compared to those currently found, see Table 7.

5 Conclusions

The algorithm of Evolutionary Strategies for the creation of minimum length addition chains has a competitive performance with respect to the length reached by other heuristic proposals.

Nevertheless, the present development it obtains in a number of operations smaller than any of the previous proposals. It is clear that using an ES can have an optimization in time with respect to the other algorithms. This is due to the fast convergence which characterizes the ES.

However this reduces the scanning ability of the algorithm which affects the quality of the proposed solutions. The above motivates to propose improvements to the algorithm developed. On the one hand a proposal can be mentioned in the aspect of the mutation used and change to another scheme that provides greater diversity. On the other hand, considering the exploration capacity, it is possible to propose another algorithm that maintains a fast convergence without losing this premise, like the Differential Evolution algorithm that has proven to be successful in this type of problems.

Acknowledgements

The authors thank the Instituto Politécnico Nacional (IPN) and the National Council of Science and Technology of Mexico (CONACyT) for the support provided for the realization of this project.

References

1. **Lenstra, H. W. (1986).** *Factoring integers with elliptic curves.* Report 86-18, Mathematisch Instituut, Universiteit van Amsterdam.
2. **Williams, H. C. (1982).** A p+l method of factoring. *Math. Comp*, Vol. 39, No. 159, pp. 225–234. DOI: 10.1090/S0025-5718-1982-0658227-7.
3. **Gordon, D. M. (1998).** A survey of fast exponentiation methods. *Journal of Algorithms*, Vol. 27, No. 1, pp. 129–146.
4. **Cruz-Cortés, N., Rodríguez-Henríquez, F., Juárez-Morales, R., & Coello-Coello, C. A. (2005)**. *Finding Optimal Addition Chains Using a Genetic Algorithm Approach.* In: **Hao, Y., Liu, J., Wang, Y.-P., Cheung, Y.-M., Yin, H., Jiao, L., Ma, J., Jiao, Y.-C. (eds.).** CIS´05, Part I. Springer, *Lecture Notes in Artificial Intelligence*, Vol. 3801, pp. 208–215. DOI:10.1007/ 11596448_30.
5. **Downey, P., Leong, B., & Sethi, R. (1981)**. Computing sequences with addition chains. *SIAM J. Computing,* Vol. 10, No. 3, pp. 638–646. DOI: 10.1137/0210047.
6. **Kaya-Koc, C. (1994).** *High-speed RSA implementation.* Technical Report, RSA Laboratories, Redwood City, CA.
7. **Kruijssen, S. V. D. (2007).** *Addition chains, efficient computing of powers.* Bachelor Proyect, Amsterdam, pp. 13–50.
8. **Kunihiro, N. & Yamamoto, H. (1998).** Window and extended window methods for addition chain and addition-subtraction chain. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, pp. 72–81.
9. **Osorio-Hernández, L. G., Mezura-Montes, E., Cruz-Cortés, N., & Rodríguez-Henríquez, F. (2009).** An improved genetic algorithm able to find minimal length addition chains for small exponents. *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1–6.
10. **León-Javier, A., Cruz-Cortés, N., Moreno-Armendáriz, M. A., & Orantes-Jiménez, S. (2009).** Finding minimal addition chains with a particle swarm optimization algorithm. *Lecture Notes in Computer Science*, Springer, 5845, pp. 680–691. DOI: 10.1007/978-3-642-05258-3_60.
11. **Nedjah, N. & Macedo, M. L. (2006).** Towards Minimal Addiction Chains Using Ant Colony Optimization. *Journal of Mathematical Modelling and Algorithms*, Vol. 5, No. 4, pp. 525–543. DOI: 10.1007/s10852-005-9024-z.
12. **Cruz-Cortés, N., Rodríguez-Henríquez, F., & Coello-Coello, C. A. (2008).** An artificial immune system heuristic for generating short addition chains. *IEEE Transactions on Evolutionary Computation*, Vol. 12, No. 1, pp. 1–24. DOI: 10.1109/TEVC.2007.906082.
13. **Domínguez-Isidro, S. & Mezura-Montes, E. (2011).** Evolutionary Programming Algorithm to Find Minimal Addition Chains. *Proceedings of the 1er. Congreso Internacional de Ingeniería Electrónica, Instrumentación y Computación.*
14. **Rechenberg, I. (1971).** Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipiender biologischen Evolution. *Frommann-Holzboog,* Vol. 15, de Reihe Problemata.
15. **Chwefel, H. P. (1975)**. *Evolutionsstrategie und numerische Optimierung.* Dissertation, Technische Universit Berlin.
16. **Correia, M. B. (2013).** A Study of Redundancy and Neutrality in Evolutionary Optimization. *Evolutionary Computation,* Vol. 21, No. 3, pp. 413–443. DOI: 10.1162/EVCO_a_00090.
17. **Yang, L. & Qing-Lan, J. (2012).** The application of improved evolutionary strategy algorithm in optimization. *Machine Learning and Cybernetics (ICMLC´12), International Conference on*, pp. 1212–1217. DOI: 10.1109/ICMLC.2012.6359528.
18. **Zubanovic, D., Hidic, A., Hajdarevic, A., Nosovic, N., & Konjicija, S. (2014).** Performance analysis of parallel master-slave Evolutionary strategies (μ,λ) model python implementation for CPU and GPU. *37th International Convention on Information and Communication Technology, Electronics and*

*Microelectronics (MIPRO´*14), pp. 1609–1613. DOI: 10.1109/MIPRO.2014.6859822.

19. **Beyer, H. & Schwefel, H., (2002).** *Evolution strategies, a comprehensive introduction.* Natural Computing Series, Springer Heidelberg, Vol. 1, No. 1, pp. 3–52. DOI: 10.1023/A:1015059928466.
20. **Rodriguez-Cristerna, A. & Torres-Jimenez, J. (2013).** A Genetic Algorithm for the Problem of Minimal Brauer Chains. *Recent Advances on Hybrid Intelligent Systems,* Springer Heidelberg. DOI: 10.1007/978-3-642-35323-9_2.
21. **Flammenkamp, A. (2016).** *Shortest Addition Chains*. wwwhomes.uni-bielefeld.de/achim_chain.html.
22. **Domínguez-Isidro, S., Mezura-Montes, E., & Osorio-Hernández, L. G. (2015).** Evolutionary programming for the length minimization of addition chains. *Engineering Applications of Artificial Intelligence,* Vol. 37, pp. 125–134. DOI:10.1016/j.engappai.2014.09.003.

Article received on 07/07/2017; accepted on 24/05/2018. Corresponding author is Mauricio Olguín Carbajal.