An Evolutionary Programming Algorithm to Find Minimal Addition Chains

S. Domínguez-Isidro and E. Mezura-Montes, *Member, IEEE*

***Abstract*-- This paper proposes an evolutionary programming algorithm (EP) to find addition chains of minimum length. Addition chains are used to reduce the number of multiplications in field exponentiation, which is used in data encryption and decryption for public-key cryptosystems. Generating minimal addition chain is considered an NP-Hard problem. Our EP begins with a series of randomly generated addition chains. EP has a mutation operator to generate new solutions and a replacement mechanism with stochastic tournaments. Our EP algorithm was tested with different types of exponents and compared with other meta-heuristic algorithms, obtaining competitive results.**

***Index Terms*--Addition chains, evolutionary programming, cryptography.**

I. Introduction

Public-key cryptosystems such as RSA, Diffie-Hellman, DSA [16]-[4]-[14]-[3] use field or modular exponentiation for data encryption and decryption. Field exponentiation consists on finding a positive integer b satisfying (1):

$$b = a^e \bmod p \qquad (1)$$

Where a is a positive integer within the range [0, 1, 2... p-1] in RSA, a is also the text to encrypt or decrypt depending the case. e is an arbitrary positive number, and p is a prime number. The problem presented in field exponentiation is the number of multiplications that involves the exponentiation operation. That is why in this work we will focus on reducing the number of multiplications required in exponentiation. For example, let $e = 27$ and a being a positive integer. To compute a^{27} we would multiply a twenty seven times, as indicated by the exponent. However to do this to encrypt and decrypt data on asymmetric encryption algorithms is computationally expensive, mostly because the exponent e generated could be larger than 128 bits [10] (~ 50 digit numbers). Therefore, an option to reduce the number of multiplications is the usage of addition chains for a given exponent, which is informally defined as a sequence of integers that satisfies the following properties:

- The first number in the chain is 1.
- Each number is generated by the addition of two previous numbers.
- The last number of the chain is the exponent e.

According with our example a^{27}, we can use the following addition chain U = [1, 2, 3, 6, 12, 13, 26, 27] with a length l = 7.

$$a^1 = a \rightarrow a^2 = a^1.a^1 \rightarrow a^3 = a^2.a^1 \rightarrow a^6 = a^3.a^3$$
$$\rightarrow a^{12} = a^6.a^6 \rightarrow a^{13} = a^{12}.a^1 \rightarrow a^{26} = a^{13}.a^{13}$$
$$a^{27} = a^{26}.a^1$$

Using the previous addition chain, we can perform exponentiation with only 7 multiplications, but for the same exponent $e=27$ we can generate another addition chain U = [1, 2, 3, 6, 9, 18, 27] with length l = 6 which is outlined as follows.

$$a^1 = a \rightarrow a^2 = a^1.a^1 \rightarrow a^3 = a^2.a^1 \rightarrow a^6 = a^3.a^3$$
$$\rightarrow a^9 = a^6.a^3 \rightarrow a^{18} = a^9.a^9 \rightarrow a^{27} = a^{18}.a^9$$

The example above shows that for a single exponent it is possible to generate several additions chains, and the one with the smallest length is better because the number of multiplications performed in the exponentiation is reduced.

There are several methods to generate addition chains of minimal length, these can be classified as deterministic [1]-[9] and stochastic [2].

Among the deterministic methods we can find the binary method [12]-[10], the factor method [12]-[11], the window method [10]-[11]-[13] and the sliding window method [10]-[11]-[13].

The stochastic methods presented here are those nature-inspired. Swarm Intelligence algorithms such as the *Ant Colony Optimization* (ACO) [6]-[15], *Particle Swarm Optimization* (PSO) [14] and the *Artificial Immune System* (AIS) [3] [5] have been used to minimize the length of addition chains. Genetic algorithms (GAs) have been also employed in [4] and in [16].

In this paper, we propose to use another evolutionary algorithm called evolutionary programming (EP) to solve this problem. Our EP algorithm was tested with a special class of

S. Domínguez-Isidro and E. Mezura-Montes are with the National Laboratory for Advanced Informatics (LANIA) A.C.. Rébsamen 80, Centro, Xalapa, Veracruz, 91000, MEXICO. Email: sdominguezi@lania.edu.mx, emezura@lania.mx

examples of up to 64 bits. Moreover, we calculated the accumulate addition chains for different sets of exponents in which competitive results were obtained in comparison with those reported in the specialized literature.

The contents of this paper are organized as follows: Section II formally states the minimal length addition chain problem. Section III introduces EP and the adapted version to deal with addition chains is presented. The experiments and results are summarized in Section IV. Finally, in Section V some conclusions are drawn and the future work is defined.

II. Problem Statement

The problem tackled in this paper is the length minimization of an addition chain U, i.e., Minimize Length (U) for a given exponent e.

An addition chain U with length computed with Length (U) is defined as a sequence of positive integers $U = u_1, u_2, ..., u_i, ..., u_l$, with $u_1 = 1$, $u_2 = 2$ and $u_l = e$, and $u_{i-1} < u_i < u_i+1 < ... < u_l$, where each u_i is obtained by adding two of the previous elements in the chain $u_i = u_j + u_k$ with $j, k < i$ *for* $i > 2$. Note that j and k are not necessarily different.

III. Proposed Algorithm

EP, as GAs, is one of the original algorithms within evolutionary computation. Any evolutionary algorithm (EA) has the following elements [7]: Solution encoding, fitness function, population, parent selection mechanism, variation operators (crossover and mutation), and survivor selection mechanism.

There are EP versions to solve numerical optimization problems as well as other search problems [8]. Unlike GAs, EP simulates the evolution at species level. Therefore, there is no crossover operator. An individual, i.e., a solution to the optimization problem, is the main element in EP and a population of individuals is considered as the starting point in the optimization process. Each individual uses a mutation operator to generate new solutions, i.e., no parent selection mechanism is used while all individuals are able to reproduce with an asexual variation operator. Therefore, EP could be considered, by design, simpler than a GA.

The survivor selection mechanism is implemented in EP to bias the search to promising regions of the search space. However, this process has random elements. The replacement is implemented by means of stochastic tournaments among the individuals in the current population and their offspring. Each individual competes in a series of binary tournaments against q individuals chosen at random from the union of the current population and their offspring. The number of wins is stored for each individual. After that, all the solutions are sorted by their number of wins and the first half is chosen to survive to the next generation and the other half is eliminated.

This survivor selection looks to promote diversity in the population because some solutions with an average fitness value are able to get a significant number of wins if the encounters are with below-average solutions. Moreover, EP requires the design of only one variation operator used to generate new solutions. The general steps of the EP algorithm are included in Algorithm 1.

The design of EP to solve the length minimization of an addition chain comprises the encoding of solutions with adequate fitness function and initial population, a mutation operator and a survivor selection mechanism. They are explained next.

A. Solution Encoding

An individual in our EP consists on an array of integer numbers representing an addition chain [16]-[14]-[3]. The fitness value of each individual is the number of elements in the array, see Fig. 1.

Algorithm 1 EP algorithm

1: Generate an initial population of individuals at random
2: Compute fitness of each individual
3: **while** a stop condition is not satisfied **do**
4: Mutate each individual in the population to generate offspring.
5: Compute fitness of each offspring.
6: Select those individuals for next generation.
7: **end while**

Our EP algorithm assumes each member in the population is feasible, i.e., it is a valid addition chain. Therefore, each addition chain in the initial population is generated by considering the following three options:

1. By applying the double stepping, that is: $u_i = 2u_{i-1}$.
2. By adding the two previous elements $u_i = u_{i-1} + u_{i-2}$.
3. By adding the last element plus a randomly chosen element $u_i = u_{i-1}+rnd(0, i-1)$ where $rnd(A,B)$ returns an integer number between A and B, generated with a uniform distribution.

Where $u_{i-1} < u_i \leq e$, i.e., the feasibility of the solutions is always maintained. As it can be noted, the first two elements within any feasible addition chain U are number 1 followed by number 2 and the third element can be 3 or 4, chosen at random. After that, the process picks, based on two parameters called f and g, among the three strategies enlisted above. The details of function Complete are in Algorithm 2, where *Flip (prob)* returns 1 with probability prob. Algorithm 2 also keeps the addition chain U from surpassing the value of exponent e.

B. Mutation Operator

The mutation operator in our EP allows each individual to generate more than one mutant [16]. Each individual U generates a mutant by choosing a mutation position at random within the addition chain, i.e., *rnd* (3, Length(U) − 1).

Id	Individual	Fitness Value
0	[1, 2, 4, 8, 10, 14, 28, 42, 46, 50, 52, 53]	l = 11
1	[1, 2, 4, 8, 16, 32, 48, 52, 53]	l = 8
2	[1, 2, 4, 6, 8, 16, 32, 48, 52, 53]	l = 9
3	[1, 2, 3, 6, 7, 14, 28, 42, 49, 52, 53]	l = 10
	...	
	...	
	...	
n	[1, 2, 3, 6, 12, 16, 32, 48, 52, 53]	l = 9

Fig. 1. Solution encoding and fitness value

Algorithm 2 Complete(*U, k, e*)
Receives an incomplete addition chain U. k is the next position to be filled and e is the exponent.
Returns a complete and feasible addition chain U

```
1: Set i = k − 1
2: while ui 6= e do
3:     if Flip(f) then
4:         ui+1 = 2ui
5:     else
6:         if Flip(g) then
7:             ui+1 = ui + ui−1
8:         else
9:             ui+1 = ui + rnd(u0, ui−1)
10:        end if
11:    end if
12:    while ui+1 > e do
13:        aux = i − 1
14:        ui+1 = ui + ui−aux
15:        aux = aux − 1
16:    end while
17: end while
18: return U
```

From this position, the remaining elements of U are erased and the function *Complete* is invoked to generate a new feasible addition chain (see Fig. 2). The process repeats t times. From the t mutants generated from U, the one with the shortest length value is selected as its offspring. Ties are broken by choosing one of the best ones at random. In others words, the mutation for our EP algorithm uses a local-search-like process to generate better offspring.

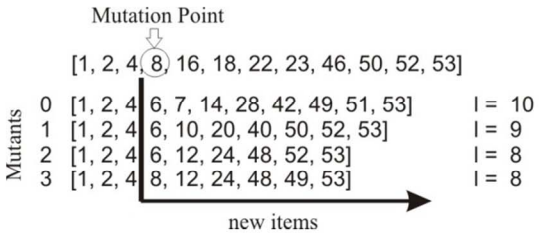

Fig. 2. Mutation operator in the proposed EP algorithm

C. Survivor selection

The stochastic tournaments to choose the individuals to remain for the next generation between the current population and their offspring are implemented by considering the way EP does it. However, in our case the length of an individual will be the comparison criterion when it competes against another one. A user-defined parameter q determines the number of opponents for all individuals in the population and it remains fixed during all the process. After all individuals have computed their wins, they are sorted based on this value and the first half will be considered the population for the next generation and the second half is eliminated, see Algorithm 3.

Algorithm 3 Survivor_Selection (*set*)
Receives a Set with the combination of the n individuals in the current population and their n offspring.
Returns the population for the next generation *Pop'* with size n.

```
1: for j = 1 to (2 * n) do
2:     wins_j = 0
3:     for k = 1 to q do
4:         i = rnd(1, 2 * n)
5:         if Length(Set_j) < Length(Set_i) then
6:             wins_j + +
7:         end if
8:     end for
9: end for
10: Sort Set based on wins values
11: for j = 1 to n do
12:    Pop'_j = Set_j
13: end for
14: return Pop'
```

The complete pseudocode for our EP approach is presented in Algorithm 4.

Algorithm 4 Our EP pseudocode

```
1: Pop= Ø
2: for j = 0 to n do
3:     Pop_j = Feasible chain(e)
4: end for
5: for k = 1 to MAXGEN do
6:     Offspring = Ø
7:     for m = 0 to n do
8:         Offspring_m = Mutation(Pop_m)
9:     end for
10:    Pop' = Replacement(Pop + Offspring)
11:    Pop=Pop'
12: end for
```

IV. Experiments and Results

We tested the performance of our EP algorithm with two experiments. The purpose of the experiments is to test the quality (the best solution reached so far) and consistency (better median value) obtained in a set of independent runs. For all experiment the following parameter values were used by the EP algorithm, which performs 240,000 evaluations:

- Population size n − 200

- Maximum number of generations *MAXGEN* = 300
- Number of mutants $t = 4$
- Number of individuals for the encounters $q = 10$
- Double stepping rate $f = 0.7$
- Previous positions rate $g = 0.2$

The first experiment consisted on calculating the total accumulated addition chains for a fixed set of "small" exponents. An accumulated addition chain (T) for a maximum value Z, represents the sum of all lengths of the addition chains obtained for all the exponents [1, 2, ..., Z], as stated in (2).

$$T(Z) = \sum_{i=1}^{z} EP(i) \quad (2)$$

A smaller T (Z) value represents a better performance by the algorithm. The following intervals for exponent e were tested [3, 14, 16]: $e \in [1, 512]$, $e \in [1, 1000]$, $e \in [1, 2000]$, $e \in [1, 2048]$, and $e \in [1, 4096]$. 30 independent runs per each exponent set were computed with the parameter values detailed above. The statistics obtained are summarized in Table I, where it can be noted that EP provided a robust performance, i.e., the best, average and median values are very close, as reflected by the low standard deviation values in the last column of the table.

In Table II we compare the best accumulated addition chains obtained by our EP algorithm against those reported by an adapted AIS [3], a GA [16], and by PSO [14].

TABLE I
EP STATISTICAL RESULTS OF THE FIRST EXPERIMENT.

$e \in$	Best	Average	Median	Worst	Std. Dev.
[1,512]	4924	4924	4924	4924	0.0000
[1,1000]	10808	10808.7	10808.5	10811	0.9487
[1,1024]	11115	11115.7	11115.5	11118	0.9487
[1,2000]	24070	24071.5	24070.5	24076	2.3214
[1,2024]	24737	24740.3	24740	24745	2.3118
[1,4096]	54454	54462.1	54463	54467	3.4140

TABLE II
COMPARISON OF BEST RESULTS FOR ACCUMULATED ADDITION AGAINST AIS [3], GA [16] AND PSO [14]. BOLDFACE MEANS THAT THE OPTIMAL VALUE WAS REACHED. ITALICS MEANS A BETTER RESULT WITH RESPECT TO THE COMPARED APPROACHES

$e \in$	Optimal	AIS	GA	PSO	EP
[1,512]	**4924**	**4924**	**4924**	---	**4924**
[1,1000]	10808	10813	10809	---	**10808**
[1,1024]	11115	11120	---	11120	**11115**
[1,2000]	24063	24108	24076	---	*24070*
[1,2024]	24731	24778	24748	---	*24737*
[1,4096]	54425	54617	54487	---	*54454*

We note for the first three sets of exponents in Table II that EP provided the optimal result. Furthermore, EP outperformed all algorithms in five out of six exponent sets.

The second experiment includes a special class of hard exponents, i.e. hard to optimize because deterministic methods are not able to obtain their minimal length addition chain [3]-[14]-[16].

TABLE III
RESULTS OF THE SECOND EXPERIMENT: BEST RESULTS FOUND BY THE GA [16] AND EP ON A SET OF "HARD" EXPONENTS.

E	Addition Chain	GA	PE
3243679	1-2-3-5-10-20-40-80-83-166-206 412-824-1648-3296-6592-9888 19776-39552-79104-158208-316416 632832-632915-1265830-2531660 3164575-3243679	27	27
3493799	1-2-3-4-8-16-32-64-128-256 512-515-1030-2060-4120-8240 16480-32960-65920-131840 263680-263683-527366-1054732 2109464-3164196-3427879-3493799	27	27
3459835	1-2-4-5-10-20-25-35-70-140-280-560 1120-2240-4480-8960-17920-35840 71680-71705-143410-286820-573640 1147280-1147305-2294610-3441915 3459835	27	27
3235007	1-2-3-6-12-24-27-54-108-162-324 648-1296-1944-3888-7776-15552 31104-62208-124416-248832-248859 497691-995382-1990764-2986146 3235005-3235007	27	27
3230591	1-2-3-4-8-16-32-64-128-131-262 524-1048-2096-4192-8384-16768 33536-67072-134144-268288-536576 538672-538803-1077606-2155212 2694015-3230591	27	27
3182555	1-2-3-6-9-18-36-72-144-288-576 1152-1728-2880-5760-11520-23040 46080-92160-184320-187200-187209 374418-748836-1497672-2995344 3182553-3182555	27	27
3440623	1-2-4-8-16-32-48-96-192-193 386772-773-1546-3092-6184 12368-24736-49472-50245-99717 199434-398868-797736-1595472 3190944-3390378-3440623	27	27
3926651	1-2-4-5-9-18-36-72-144-288 576-1152-2304-4608-9216-18432 18437-36869-73738-92175-184350 368700-737400-811138-1548538 3097076-3908214-3926651	27	27
3234263	1-2-3-6-12-24-48-96-192-194 388-776-1552-3104-3107-3155 6310-12620-25240-50480-100960 201920-403840-807680-1615360 3230720-3233875-3234263	27	27
3352927	1-2-4-8-16-17-34-68-136-272 544-1088-2176-4352-8704-17408 34816-69632-139264-278528 557056-1114112-1114656-1114724 1114741-2229482-3344223-3352927	27	27

A comparison based on the best results found out of 30 independent runs per exponent by the EP against those reported by the GA in [16] is presented in Table III. The same parameters for EP used in the first experiment were also employed in this second experiment. The other two algorithms used in the previous experiment did not report the solution of

such exponents. As it can be noted, EP found the same results with respect to those of the GA. It is important to remark that PE is easier to implement than the GA in [16] because the last one requires two variation operators (crossover and mutation), the use of elitism and a parent selection mechanism.

V. Conclusions and Future Work

In this paper we proposed an EP algorithm to find addition chains of minimal length. The core of the EP algorithm comprises the solution encoding with suitable fitness function and initial population, a mutation operator, and the survivor selection mechanism. These elements are easy to implement and EP does not uses other operators such as crossover nor additional mechanisms like parent selection in GAs [16]. Our algorithm was able to generate better accumulated addition chains with respect to three state-of-the-art algorithms (AIS [3], GA[16] and PSO[14]). Further results showed that EP was also competitive in a set of hard exponents with respect to a GA [16].

As future work, we will analyze the parameters required by the algorithm to reduce the number of evaluations without affecting its efficiency. Also, the algorithm will be adapted to solve larger exponents (more than 160 bits).

VI. Acknowledgments

The first author acknowledges support from CONACyT through a scholarship to pursue graduate studies at LANIA. The second author acknowledges support from CONACyT through project number 79809.

VII. References

[1] F. Bergeron, J. Berstel, and S. Berlek. “Efficient computation of addition chains”. *Journal de Theorie des Nombres de Bordeaux*, Francia., tome 6, no. 1:2–10, 1994.

[2] J. Bos and M. Coster. “Addition chains heuristics”. Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdan, The Netherlands., *Springer-Verlag*:1–5, 1998.

[3] N. Cruz-Cortés, F. Rodríguez-Henríquez, and C. A. Coello-Coello. “An artificial immune system heuristic for generating short addition chains”. *IEEE Transactions on Evolutionary Computation*, 12(1):1–24, February 2008.

[4] N. Cruz-Cortés, F. Rodríguez-Henríquez, R. Juárez-Morales, and C. A. Coello-Coello. “Finding optimal addition chains using a genetic algorithm approach”. *Lecture Notes in Computer Science*, Computer Science Section, Electrical Engineering Department, CINVESTAV IPN, 2:1–8, 2005.

[5] L. N. de Castro and J. Timmis. Artificial Immune Systems: “A New Computational Intelligence Approach”. *Springer Verlag*, 2002.

[6] M. Dorigo, V. Maniezzo, and A. Colorni. “The Ant System: Optimization by a Colony of Cooperating Agents”. *IEEE Transactions of Systems, Man and Cybernetics-Part B*, 26(1):29–41, 1996.

[7] A. Eiben and J. E. Smith. “Introduction to Evolutionary Computing”. *Natural Computing Series. Springer Verlag*, 2003.

[8] L. J. Fogel. Intelligence Through Simulated Evolution. “Forty years of Evolutionary Programming”. *John Wiley & Sons*, New York, 1999.

[9] D. M. Gordon. “A survey of fast exponentiation methods”. Technical report, Center for Communications Research, Westerra Court, San Diego, CA., 1997.

[10] C. Kaya-Koc. “High-speed RSA implementation”. Technical report, RSA Laboratories, Redwood City, CA, 1994.

[11] D. E. Knuth. “The art of computer programming”. *Addision-Wesley, second edition,* 1981.

[12] S. V.-D. Kruijssen. “Addition chains, efficient computing of powers”. *Bachelor Proyect*, Amsterdam, 1:13–50, 2007.

[13] N. Kunihiro and H. Yamamoto. “Window and extended window methods for addition chain and addition-subtraction chain”. *Special Section on Cryptography and Information Security*, 8:60 – 61, Enero 1998. IEICE Trans Fundamentals.

[14] A. León-Javier, N. Cruz-Cortés, M. A. Moreno-Armendériz, and S. Orantes-Jiménez. “Finding minimal addition chains with a particle swarm optimization algorithm”. *Lecture Notes in Computer Science*, 5845/2009:680–691, 2009.

[15] N. Nedjah and L. de Macedo-Mourelle. “Finding minimal addition chains using ant colony”. *In Proceedings of the international conference on intelligent data engineering and automated learning*, pages 1–6. Springer, 2004.

[16] L. G. Osorio-Hernández, E. Mezura-Montes, N. Cruz-Cortés, and F. Rodríguez-Henríquez. “An improved genetic algorithm able to find minimal length addition chains for small exponents”. *In Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1–6. IEEE Press, 2009.

VIII. Biographies

Saúl Domínguez Isidro was born in Las Choapas city Veracruz, on September 16, 1985. He holds a BsC on Informatics from UNAM and Sotavento University. He is currently pursuing a MsC in Applied Computing at the National Laboratory for Advanced Informatics (LANIA) A.C.. His research interests are Artificial intelligence, especially nature-inspired algorithms, cryptography and expert systems.

Efrén Mezura-Montes is a full-time researcher at the National Laboratory for Advanced Informatics (LANIA) A.C. in Xalapa, Veracruz, MEXICO. His research interests are the design, analysis and application of bio-inspired algorithms to solve complex optimization problems. He has published over 40 papers in peer-reviewed journals and conferences. He also has one edited book published by Springer and six book chapters published by international publishing companies.