Addition Chain Length Minimization with Evolutionary Programming

Saúl Domínguez-Isidro, Efrén Mezura-Montes and Luis G. Osorio-Hernández
Laboratorio Nacional de Informática Avanzada (LANIA) A.C.
Rébsamen 80, Centro, Xalapa, Veracruz, 91000, MEXICO
sdominguezi@lania.edu.mx, emezura@lania.mx, luisgosher@gmail.com

ABSTRACT

This paper presents the use of an evolutionary metaheuristic algorithm called evolutionary programming to minimize the length of addition chains, which is an NP-hard problem. Addition chains are used in modular exponentiation for data encryption and decryption public-key cryptosystems, such as RSA, DSA and others. The algorithm starts with a population of feasible addition chains. After that, the combination of a mutation operator, which allows each individual to generate a feasible offspring, and a replacement process based on stochastic encounters provides a simple approach which is tested on exponents with different features. The proposed algorithm is able to find competitive results with respect to other nature-inspired metaheuristic approaches but with a lower number of evaluations per run.

Categories and Subject Descriptors

I.2.8 [**Computing Methodologies**]: Problem Solving, Control Methods, and Search—*Heuristic methods*; E.3 [**Data**]: Data Encryption

General Terms

Algorithms

Keywords

Addition Chains, Evolutionary Programming, Cryptography

1. INTRODUCTION

Field exponentiation consists in finding a positive integer b satisfying equation: $b \equiv a^e \ mod \ p$ where a is a integer within the range [0, 1, 2, ..., p - 1], e is an arbitrary positive number, and p is a prime number. Field exponentiation has an inherent problem of the general exponentiation, which consists on multiplying the base a by itself as many times as specified by the exponent e. One way to reduce the computational cost involved is based on reducing the number of multiplications by using addition chains.

An addition chain U for a given exponent e, and with length denoted as $Length(U)$, is a sequence of integers with the following properties: the first number is one; every subsequent number is the sum of two early numbers not necessarily different), and e occurs at the end of chain.

This work introduces an approach based on evolutionary programming (EP) to find minimal length addition chains for different types of exponents.

GECCO'11, July 12–16, 2011, Dublin, Ireland.
ACM 978-1-4503-0690-4/11/07.

2. EVOLUTIONARY PROGRAMMING

EP simulates evolution at species level. Therefore, no crossover operator is employed. In this proposed approach, an individual is represented at genotype level, i.e., an individual is a feasible addition chain. The *fitness value* of each individual is the length of the addition chain. Therefore, shorter strings are preferred. To generate the *initial population* each individual $U_m = u_1, u_2, \ldots, e$ starts with $u_1 = 1,\ u_2 = 2,\ u_3 = rnd(3, 4)$. After that, function *Complete* (see Alg.1) is invoked to complete the addition chain.

Algorithm 1 Complete(U, k, e) receives an incomplete addition chain U. k is the next position to be filled and e is the exponent. The function returns a complete and feasible addition chain U. *Flip*(P) returns 1 with probability P.

```
1: Set i = k − 1
2: while u_i ≠ e do
3:    if Flip(f) then
4:       u_{i+1} = 2u_i {/*applying the double stepping*/}
5:    else
6:       if Flip(g) then
7:          u_{i+1} = u_i+u_{i−1} {/*adding the two previous elements*/}
8:       else
9:          u_{i+1} = u_i + rnd(u_0, u_{i−1}) {/*adding the last element
            plus a randomly chosen element*/}
10:      end if
11:   end if
12:   while u_{i+1} > e do
13:      aux = i − 1
14:      u_{i+1} = u_i + u_{i−aux}
15:      aux = aux − 1
16:   end while
17: end while
18: return U
```

The while loop observed in row 12 within Algorithm 1 is added so as to keep the addition chain U from surpassing the value of exponent e. Unlike other options proposed in [1, 2, 3], where random values are considered, in this work the search for the feasible term starts in a deterministic way from the previous position u_{i-1} stored in a variable called aux.

Mutation is used as the only variation operator. Each individual k in the current population generates t mutants, see Fig.1, and the best of them is chosen as k's offspring. To generate mutants Alg.1, is invoked t times for each individual k.

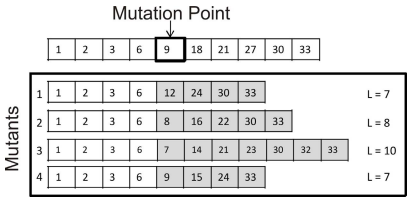

Figure 1: Mutation operator in the proposed EP algorithm

The *replacement mechanism*, as in the original EP, is carried out by stochastic encounters. Each individual, from the set of current solutions plus their corresponding offspring, competes, based on fitness, against q randomly chosen individuals from that set in head-to-head encounters. After that, the individuals are sorted based on their number of wins and the first half remains as the population for the next generation, while the rest is eliminated from the optimization process.

3. EXPERIMENTS AND RESULTS

Four experiments were designed to test the performance of the EP algorithm. "Small", "hard" and "diverse" exponents (up to 64 bits) were used with the goal to test EP in different search spaces. Quality (the best solution reached so far) and consistency (better mean values) obtained in a set of independent runs were considered as performance criteria. The following parameter values were used by the EP algorithm: population size $n = 100$, maximum number of generations MAXGEN = 230, number of mutants $t = 4$, number of individuals for the encounters $q = 10$, double stepping rate $f = 0.7$, previous positions rate $g = 0.2$.

Based on the aforementioned parameter values, the EP algorithm computed $92,000$ evaluations per execution ($n \times MAXGEN \times t$). 30 independent runs per each experiment were carried out.

The first experiment consisted in 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 Equation 1.

$$T(Z) = \sum_{i=1}^{Z} EP\_Optimal\_Addition\_Chain\_Found(i) \quad (1)$$

The comparison of those shortest addition chains (best results) found, besides the results of 95%-confidence unpaired two sample t-tests between each one of the three compared algorithm and EP are presented in Table 1, where it can be noted that EP provided a highly competitive performance. Another advantage of EP with respect to the GA [3] and PSO [2] is that the proposed algorithm required $92,000$ evaluations in each single run, while the GA and the PSO computed $240,000$ and $300,000$, respectively. The AIS algorithm did not report the evaluations per run performed.

The second experiment included a set of exponents known as "hard" to optimize because deterministic methods fail to obtain their minimal length addition chain [1, 2, 3]. In the third experiment, another set of 28 "diverse" exponents were solved by the EP algorithm and the results were compared with those obtained by the modified AIS [1] and PSO [2]. The fourth experiment is similar to the first one. However, the number of evaluations per run by EP was only $25,000$ Due to space restrictions the results are not shown. However, it can be said that EP was able to provide similar or even better results by requiring less evaluations with respect to the algorithms compared [1, 2, 3].

Table 1: Results for Accumulated addition chains by AIS [1], GA [3], PSO [2] and EP. "(+)" means a significant difference with respect to EP

$e \in$	Opt.	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*(+)	—	*24076*
[1,2048]	24731	24778(+)	24748(+)	—	*24745*
[1,4096]	54425	54617(+)	*54487*(+)	—	54497

4. CONCLUSIONS AND FUTURE WORK

The use of evolutionary programming to solve the minimal length addition chain problem was presented. The algorithm is based only on a mutation operator and a stochastic replacement process to bias the search to competitive solutions by requiring less evaluations per run with respect to some state-of-the-art nature-inspired algorithms. Four experiments with different types of exponents were carried out. Regarding the accumulated addition chains for "small" exponents, EP was able to provide competitive and even better results with only 30% of the evaluations required per run by two state-of-the-art algorithms. This behavior was also observed in two sets, one of "hard" and another of "diverse" exponents where the EP algorithm obtained similar best results with respect to the three compared algorithms [1, 2, 3], all of them with a lower number of evaluations. Finally, the EP algorithm could provide results close to those reported in experiment 1 by only requiring $25,000$ evaluations per run.

The future work consists on analyzing the parameters required by the algorithm (n, t and q). Moreover, the algorithm will be adapted to solve large exponents (more than 160 bits).

5. ACKNOWLEDGMENTS

The authors acknowledge support from CONACyT project 79809.

6. REFERENCES

[1] 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.

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

[3] L. G. Osorio-Hernández, E. Mezura-Montes, N. Cruz-Cortés, and F. Rodríguez-Henríquez. An improved genetic algorithm able to find minimal lenght addition chains for small exponents. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1422–1429. IEEE Press, 2009.