On Finding Short Addition Chains for Large Integers

Xiaopeng Zhao[1(✉)], Zhusen Liu[2] and Jiawei Qian[3]

[1] School of Computer Science and Technology, Donghua University, Shanghai 201620, China
zxp@dhu.edu.cn
[2] Zhejiang Lab, Hangzhou, Zhejiang 311121, China
liuzs@zhejianglab.com
[3] School of Information Management, Shanghai Lixin Institute of Accounting and Finance,
Shanghai 201209, China
jwqian@stu.ecnu.edu.cn

Abstract. The addition chain for a given exponent n is an increasing sequence of positive integers: its first term is 1, and each subsequent term is obtained by adding the previous two terms (which can be the same), so that the last element of the sequence is equal to n. Constructing the shortest addition chain for a fixed exponent n is the most efficient method for computing x^n in some group under multiplication. Therefore, the addition chain plays a crucial role in modern cryptography. It can improve the computational efficiency of cryptographic algorithms that require fast exponentiation, such as RSA, ElGamal, Paillier, and ECC, etc. However, the problem of finding the shortest additive chain is **NP**-Complete. Moreover, the existing evolutionary algorithms cannot work well for finding short addition chains for large integers. This paper integrates genetic algorithms with the window method to obtain an efficient strategy for the addition chain problem involving large integers. We do experiments on an RSA-1536 modulus to verify the efficiency and practicability of our algorithm.

Keywords: Addition chains · Genetic algorithms · Window method · Exponent iation · Cryptography

1 Introduction

Exponentiation is the most fundamental operation in modern cryptography. Various cryptographic algorithms, such as RSA, ElGamal, Paillier, and ECC, require the computation of power exponentiation, i.e., modular exponentiation and scalar multiplication. Constructing the shortest addition chain for a fixed exponent n is the most efficient method for computing x^n in some group under multiplication. Therefore, the efficient implementation of modern cryptographic systems cannot be separated from the design of the shortest addition chain. This highlights the significant practical implications of research on addition chains. On the other hand, an addition chain itself is also an interesting mathematical research object, and the exploration of the shortest addition chain also holds great theoretical significance. First, we give the formal definition of an additive chain:

Definition 1. *An* additive chain *of length r for a positive integer n is a sequence of positive integers as follows:*

$$1 = a_0, a_1, a_2, \dots, a_r = n;$$

Among them, for each $i = 1,2, \dots, r$, there exist positive integers j, k satisfying

$$a_i = a_j + a_k \quad k \le j < i. \tag{1}$$

If for each i in (1) we have $j = i - 1$, we call this addition chain a star chain.

For example, in volume 2 of Knuth's book, "The Art of Computer Programming" [1], an example is given where $1 \to 2 \to 3 \to 6 \to 7 \to 14 \to 15$ is a 6-length addition chain, while the shortest addition chain for 15 is of length 5, e.g., $1 \to 2 \to 3 \to 6 \to 12 \to 15$. We use $\ell(n)$ to represent the length of the shortest addition chain for n.

The problem of determining the exact value of $\ell(n)$ was first proposed by H. Dellac in 1894. In the same year, E. de Jonquières [2] introduced a factorization method for constructing addition chains, which tells us that

$$\ell(mn) \le \ell(m) + \ell(n)$$

Afterward, numerous researchers conducted extensive studies on asymptotic bounds of $\ell(n)$. One of the well-known conjectures about the lower bound of $\ell(n)$ is

$$\ell(n) \ge \lfloor \log_2(n) \rfloor + \lceil \log_2(\mu(n)) \rceil, \tag{2}$$

where $\mu(n)$ represents the number of 1s in the binary representation of n. Although A. Schönhage [3] has proved that

$$\ell(n) \ge \left\lceil \log_2(n) + \log_2(\mu(n)) - 2.13 \right\rceil$$

in 1975, improving this result is very challenging. In this paper, we use (2) as a standard to measure the quality of the resulting additive chain of n. It is worth mentioning that the famous Scholz-Brauer conjecture

$$\ell(2^n - 1) \le n + \ell(n) - 1.$$

remains unresolved. This conjecture is related to the construction of the shortest star chain. For the study of the upper bound, as early as 1939, A. Brauer [4] gave an upper bound for $\ell(n)$, and later, Yao [5] made corresponding improvement to Brauer's result.

The problem of finding the shortest addition chain is an **NP**-Complete problem [6]. In the last century, research on additive chains has produced dozens of algorithms, including binary methods [1], Brauer's method [4], window or m-ary methods [1], Bos-Coster heuristics [7], factoring methods [1], continued fraction/Euclid methods [8], Knuth's power tree methods [1], genetic algorithms [9,10], BGMW algorithms [11], Yacobi's data compression methods [12], Bocharova-Kudryashov's data compression methods [13] and so on. In 2016, Clift [14] calculated the shortest additive chain for all positive integers less than 2^{36}. Later, Clift proved that $\ell(2^n - 1) = n + \ell(n) - 1$ for any $\ell(n) \leq 8$ (see [14]).

1.1 Related Work

The window or m-ary method was proposed in [1]. In 1989, Bos and Coster [7] described four heuristic methods for making an addition sequence of a set of numbers, namely Approximation, Division, Halving and Lucas. However, the report shows that the weight function in each method does not give really satisfactory results. Recently, Ding et al. [15] introduced a cross window method and its variant for improving the window method.

In 2016, Picek et al. [9, 10] proposed a genetic algorithm with new crossover and mutation operators to shorten the length of the addition chains for a given exponent. They also investigated values up to $2^{255} - 21$ and the results indicate that the proposed approach is a very promising alternative to deal with this problem.

1.2 Our Contributions

According to our experiments, the existing genetic algorithms for finding short addition chains are not good at handling (extremely) large integers, such as the commonly used RSA moduli in cryptography. In this paper, we integrate genetic algorithms with the window method to obtain an efficient strategy for the addition chain problem involving (extremely) large integers. Specifically, on the basis of tuned genetic algorithm, we give the implementation of MakeSequence algorithm, which is to construct an addition chain containing a specific set of integers. Through the experiment of an RSA-1536 modulus, we confirm the efficiency and practicability of our proposed algorithm.

2 Preliminaries

In this section, we briefly introduce the window method and the genetic algorithm for finding short addition chains. These two methods will be important in the study of our proposed algorithm in Section 3.

2.1 Window Method

The window or m-ary method was proposed in [1]. The basic idea of this method is to represent the large integer in binary and split it into pieces, called windows. When the window size is equal to 1, it will degenerate into the binary method. Take the large integer 26235947428953663183191 as an example, and set the window size to 5. Consider splitting its binary representation of length 75 as

$$\underbrace{\underline{1011}}_{11}\,000\,\underbrace{\underline{111}}_{7}\,00\,\underbrace{\underline{1}}_{1}\,000000\,\underbrace{\underline{11101}}_{29}\,00\,\underbrace{\underline{101}}_{5}\,00\,\underbrace{\underline{11101}}_{29}\,0\,\underbrace{\underline{1}}_{1}\,000000\,\underbrace{\underline{10111}}_{23}\,\underbrace{\underline{1}}_{1}$$
$$00000\,\underbrace{\underline{11111}}_{31}\,00\,\underbrace{\underline{11001}}_{25}\,0\,\underbrace{\underline{10101}}_{21}\,\underbrace{\underline{11}}_{3}$$

If we have constructed an addition chain of length 12 containing all the above non-zero windows, e.g.,

$$1 \to 2 \to \underline{3} \to 4 \to \underline{5} \to \underline{7} \to \underline{11} \to 16 \to \underline{21} \to \underline{23} \to \underline{25} \to \underline{29} \to \underline{31},$$

then we can take the first window $\underline{1101}$ and repeatedly square it. For each window, we only need to make one addition to put it into place. Thus, the total length of the resulting addition chain is $12 + 71 + 13 - 1 = 95$.

2.2 Genetic Algorithm

In this section, we introduce the genetic algorithm for constructing addition chains proposed by Picek et al. [9] (with minor changes according to our experiments). Figure 1 shows the outline. The initialization algorithm (see Algorithm 1) generates an additive chain through random summation. The tail element is highly likely to be selected with the aim of quickly reaching the vicinity of n. Meanwhile, this algorithm also offers abundant diversity. By calling the initialization algorithm multiple times, we obtain the initial population. The fitness function for an addition chain is chosen to be its length.

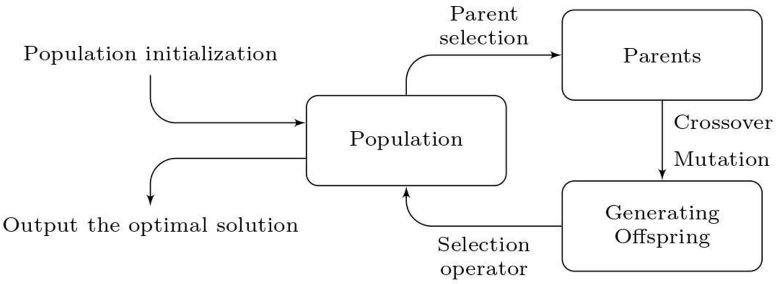

Fig. 1. Outline of genetic algorithm

Algorithm 1 Initialization algorithm

Input: A positive integer n

Output: An addition chain $e = \{e_0, e_1, \ldots, e_k = n\}$ for n

```
Set $e_0 = 1$, $e_1 = 2$, and $e_2$ to 3 or 4 uniformly at random.
Double the tail element with probability $10\%$, until it exceeds $n/2$.
while the tail element of $e$ is not equal to $n$ do
    Randomly choose among the following operations to get the next element and
    ap pend it to $e$, provided that it does not exceed $n$.
        (1) Add two randomly chosen elements from $e$.
```

```
            (2) Add the tail element to a randomly chosen element from $e$.
            (3) Loop forward from the tail element until you find the largest element
        that can be added with the tail element.
            (4) Find two elements in $e$ such that their sum is $n$.
end while
return $e$
```

The pseudocode for the one-point crossover operator is provided in Algorithm 2. It is expected that this crossover operator will produce offspring with a shorter length. Figure 2 illustrates the execution process of the crossover operator with $n = 15$ and $rand = 3$. The pseudocode for the mutation operator is shown in Algorithm 3.

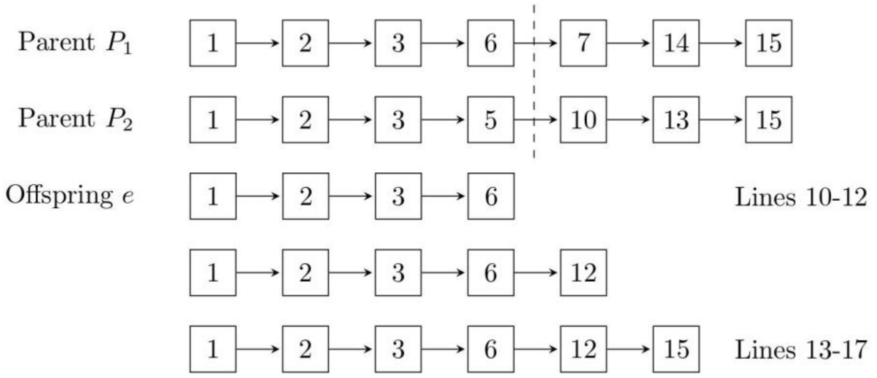

Fig. 2. An Example of Algorithm 2

Algorithm 2 Crossover operator

Input: A positive integer n and parent addition chains P_1, P_2 for n
Output: Offspring addition chain $e = \{e_0, e_1, \ldots, e_k = n\}$

```
1: procedure $RepairChain(e, n)$
2:     Delete duplicate elements in the chain.
3:     Delete elements larger than $n$ in the chain.
4:     Check whether the elements are arranged in ascending order, if not, sort them.
5:     while the tail element of $e$ is not equal to $n$ do
6:         Randomly choose among the following operations to get the next element and
           append it to $e$, provided that it does not exceed $n$.
               (1) Add two randomly chosen elements from $e$.
               (2) Add the tail element to a randomly chosen element from $e$
```

```
            (3) Loop forward from the tail element until you find the largest element that can be added with the tail element.
            (4) Find two elements in $e$ such that their sum is $n$.
        end while
    end procedure
    $rand \leftarrow random(3, k-1)$
    for $0 \le i \le rand$ do
        $e_i = P_{1_i}$
    end for
    for $rand + 1 \le i \le k$ do
        Find the lexicographically smallest pair $(pair_1, pair_2)$ such that $P_{2_i} = P_{2_{pair_1}} + P_{2_{pair_2}}$ and $e_{pair_1} + e_{pair_2} \le n$. If it does not exist, break the inner *for* loop.
        $e_i = e_{pair_1} + e_{pair_2}$
    end for
    $e \leftarrow RepairChain(e, n)$
    return $e$
```

Algorithm 3 Mutation operator

```
Input: A positive integer $n$ and an addition chain $e = \{e_0, e_1, \dots, e_k = n\}$ for $n$
Output: A Mutation addition chain $e' = \{e'_0, e'_1, \dots, e'_k = n\}$
$rand \leftarrow random(2, k-1)$
$rand_2 \leftarrow random(0,1)$
if $rand_2 == 1$ then
    $e_{rand} = e_{rand-1} + e_{rand-2}$
else
    $rand_3 \leftarrow random(2, rand-1)$
    $e_{rand} = e_{rand-1} + e_{rand_3}$
end if
$e' \leftarrow RepairChain(e, n)$
return $e$
```

3 Finding Short Addition Chains for Extremely Large Integers

Although the genetic algorithm described in Section 2.2 can quickly generate short addition chains for a wide set of exponent sizes, our experiments show that it is challenging to obtain satisfactory solutions for (extremely) large integers, such as the commonly used RSA moduli in cryptography, and the algorithm takes a notably long time. The main reason is that the solution space of the problem is too extensive, making it difficult for genetic algorithms to converge to a global optimal or satisfactory solution. In view of this, we shall integrate genetic algorithms with the window method described in Section 2.1 to obtain an efficient strategy for the addition chain problem involving large integers.

The key to solving the addition chain problem using the window method is to construct an addition chain containing a specific set of integers, known as the MakeSequence algorithm. Algorithm 4 presents our design based on the genetic algorithm in Section 2.2.

Algorithm 4 MakeSequence

```
Input: Positive integers $a_1 < a_2 < \cdots < a_k$
Output: An addition chain containing $a_1, \ldots, a_k$
Call the genetic algorithm in Section 2.2 to generate multiple short addition chains
  for $a_1$, and put the top $r_1$ addition chains into the bucket $B_1$.
for $2 < i < \mathrm{k}$ do
    For each addition chain $e$ in $B_{i-1}$, replace $e$ with $RepairChain(e, a_i)$.
    On the basis of the initial population $B_{i-1}$, call the genetic algorithm in Section
      2.2 to generate multiple short addition chains for $a_i$ containing $a_1, \ldots, a_{i-1}$.
    Put the top $r_i$ optimal individuals into the bucket $B_i$.
end for
Output the individual with the shortest length in bucket $B_k$.
```

4 Experimental Results

All the following experiments are conducted on a personal computer equipped with an Intel i5 processor (2.30GHz), 8GB RAM, and Windows 10 OS. All the algorithms introduced earlier are coded in C++, and the development environment is Visual Studio 2017. The GMP library is used for large number operations, and its version is 6.0.0. The specific parameter settings are as follows: the number of generations is 600, the initial population size is 2000, the number of crossovers is 500 per generation, and the number of mutations is 10 per generation.

To test the effectiveness of the genetic algorithm described in Section 2.2, we evaluate some of the datasets in [10], as shown in Table 1. The orange font indicates that the genetic algorithm has found the optimal solution for this test, which is the shortest addition chain. The “difficult” value $n = 2^{255} - 21$ is a commonly used parameter for implementing elliptic curve cryptography algorithms. For this value, approximately 50 independent runs can obtain an addition chain of length 267 (slightly better than the result given in [10]).

Table 1. Exponents up to $2^{255} - 21$

n	$\ell(n)$	Binary method	Window method	Genetic Algorithm	
			(optimal)	Optimal	Average
488705	23	26	27	**23**	23.53
1273909	25	29	30	**25**	25.87
3399779	25	31	31	**25**	26.87
5425679	27	32	32	28	28.23
9264263	28	34	34	**28**	29.63
20279147	29	39	36	30	31.07
51950083	30	34	36	**30**	31.20
115216741	31	39	38	32	33.60
159963579	33	41	39	34	35.20
310469637	33	36	39	34	34.23
1073740801	35	49	41	**35**	35.26
17182318319	N/A	49	46	42	46.77
$2^{127} - 3$	N/A	251	163	136	141.49
$2^{255} - 21$	N/A	506	323	267	277.25

Next, we use an RSA-1536 modulus (the prime factorization of n is unknown)

$n =$
184769970321174147430683562020016440301854933866341017147178577491065169671116124985933768430543574458561606154457179405222
9

717732524660960646946071249623720442022269756756687378427562
38

950876467844093328515749657884341508847552829818672645133986
3

649319080846719904318743812833635027954702826532978029349161
5

811881049844908319545009483937752272570525785919449938700736
9

755688436933812779613089230325696952532616208236764903160365
51

37144791393234716956698806
9

to test the effectiveness of the algorithm proposed in Section 3. The MakeSequence algorithm we designed has a very good effect. We use the window method with size 15 to solve the problem and construct an addition chain of length 170 that contains 95 windows (with a maximum value of 32151) by Algorithm 4. The length of the addition chain we constructed is much smaller than the upper bound given by Yao [5]:

$$\ell(a_1, a_2, \cdots, a_{95}) \leq \log_2(32151) + \left(2 + \frac{4}{\sqrt{\log_2(32151)}}\right)\frac{95 \cdot \log_2(32151)}{\log_2\left(\log_2(32151)\right)}$$
$$\approx 1120$$

In about two hours, we find the resulting addition chain of length 1786. which is available at https://pan.baidu.com/s/1Qjyjsg8VUY8aE7b6HQ3J6Q (password: e2m5).

5 Conclusion and Future Research

In this paper, we give an effective method for finding short addition chains for (extremely) large integers. This method is based on the organic combination of the window method and genetic algorithm. Through the experiment of an RSA-1536 modulus, we confirm the efficiency and practicability of our proposed algorithm. In fact, the bottleneck of our method is how short the addition chain can be found by the optimal window method. Therefore, the future work is to study how to apply genetic algorithm to other extended window methods, such as the cross window method in [15].

Acknowledgments. We are grateful to the referee for carefully reading our manuscript and for his/her valuable comments. This work was supported in part by the Shanghai Science and Technology Commission 23YF1401000, in part by the Fundamental Research Funds for the Central Universities under Grant 2232022D-25, and in part by the China Postdoctoral Science Foundation (No. 2023M733265).

References

1. Donald, E.K.: The Art of Computer Programming, vol. 2: Seminumerical Algorithms. Addison-Wesley, Reading (1981)
2. De Jonquieres, E.: Question 49 (h. dellac). L'Intermédiaire Math **1**(20), 162–164 (1894)
3. Schönhage, A.: A lower bound for the length of addition chains. Theoretical Computer Science **1**(1), 1–12 (1975)
4. Brauer, A.: On addition chains. Bulletin of the American mathematical Society **45**(10), 736–739 (1939)
5. Yao, A.C.-C.: On the evaluation of powers. SIAM J. Comput. **5**(1), 100–103 (1976)
6. Downey, P.J., Leong, B.L., Sethi, R.: Computing sequences with addition chains. SIAM J. Comput. **10**(3), 638–646 (1981) https://doi.org/10.1137/0210047
7. Bos, J.N., Coster, M.J.: Addition chain heuristics. In: Proc. Advances in Cryptology - CRYPTO '89. Lecture Notes in Computer Science, vol. 435, pp. 400–407. Springer, Heidelberg (1989). https://doi.org/10.1007/0-387-34805-0_37
8. Brlek, S., Castéran, P., Habsieger, L., Mallette, R.: On-line evaluation of powers using Euclid's algorithm. RAIRO-Theoretical Informatics and Applications **29**(5), 431–450 (1995)
9. Picek, S., Coello, C.A.C., Jakobovic, D., Mentens, N.: Evolutionary algorithms for finding short addition chains: Going the distance. In: Proc. 16th European Conference - EvoCOP

2016. Lecture Notes in Computer Science, vol. 9595, pp. 121–137. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30698-8_9

10. Picek, S., Coello, C.A.C., Jakobovic, D., Mentens, N.: Finding short and implementation-friendly addition chains with evolutionary algorithms. Journal of Heuristics **24**(3), 457–481 (2018)
11. Brickell, E.F., Gordon, D.M., McCurley, K.S., Wilson, D.B.: Fast exponentiation with precomputation (extended abstract). In: Proc. Advances in Cryptology EUROCRYPT '92. Lecture Notes in Computer Science, vol. 658, pp. 200–207. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-47555-9_18
12. Yacobi, Y.: Fast exponentiation using data compression. SIAM Journal on Computing **28**(2), 700–703 (1998)
13. Kunihiro, N., Yamamoto, H.: Window and extended window methods for addition chain and addition-subtraction chain. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences **81**(1), 72–81 (1998)
14. http://wwwhomes.uni-bielefeld.de/achim/addition chain.html
15. Ding, Y., Guo, H., Guan, Y., Song, H., Zhang, X., Liu, J.: Some new methods to generate short addition chains. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(2), 270–285 (2023) https://doi.org/10.46586/TCHES.V2023.I2.270-285