Finding Optimal Addition Chains Using a Genetic Algorithm Approach

Nareli Cruz-Cortés, Francisco Rodríguez-Henríquez,
Raúl Juárez-Morales, and Carlos A. Coello Coello

Computer Science Section, Electrical Engineering Department,
Centro de Investigación y de Estudios Avanzados del IPN,
Av. Instituto Politécnico Nacional No. 2508, México D.F.
nareli@computacion.cs.cinvestav.mx,
{francisco, ccoello}@cs.cinvestav.mx

Abstract. Since most public key cryptosystem primitives require the computation of modular exponentiation as their main building block, the problem of performing modular exponentiation efficiently has received considerable attention over the years. It is known that optimal (shortest) addition chains are the key mathematical concept for accomplishing modular exponentiations optimally. However, finding an optimal addition chain of length r is an **NP**-hard problem whose search space size is comparable to $r!$. In this contribution we explore the usage of a Genetic Algorithm (GA) approach for the problem of finding optimal (shortest) addition chains. We explain our GA strategy in detail reporting several promising experimental results that suggest that evolutionary algorithms may be a viable alternative to solve this illustrious problem in a quasi optimal fashion.

1 Introduction

Arguably, the field or modular exponentiation is the most important single arithmetic operation in public key cryptosystems. The search for efficient algorithm solutions for this problem has a long history whose roots can be traced as far back as the ancient works of Indian mathematicians in 200 B.C [1]. In addition to its historical and theoretical relevance, field exponentiation has many important practical applications in the areas of error-correcting codes and cryptography. Modular exponentiation is used in several major public-key cryptosystems such as RSA, Diffie-Hellman and DSA [2]. For instance, the RSA crypto-scheme is based on the computation of $M^e \bmod n$, where e is a fixed number, M is an arbitrarily chosen numeric message and n is the product of two large primes, namely, $n = pq$. Typical bit-lengths for n used in commercial applications range from 1024 up to 4096 bits. In addition, modular exponentiation is also a major building block for several number theory problems including integer prime testing, integer factorization, field multiplicative inverse computation, etc.

Let α be an arbitrary integer in the range $[1, n-1]$, and e an arbitrary positive integer. Then, we define modular exponentiation as the problem of finding the

Y. Hao et al. (Eds.): CIS 2005, Part I, LNAI 3801, pp. 208–215, 2005.
© Springer-Verlag Berlin Heidelberg 2005

unique integer $\beta \in [1, n-1]$ that satisfies the equation

$$\beta = \alpha^e \bmod n \tag{1}$$

In order to improve legibility, in the rest of this paper we will drop the modular operator whenever it results unambiguous.

The problem of determining the correct sequence of multiplications required for performing a modular exponentiation can be elegantly formulated by using the concept of *addition chains*. Formally, An *addition chain* for e of length l is a sequence U of positive integers, $u_0 = 1, u_1 \ldots, u_l = e$ such that for each $i > 1$, $u_i = u_j + u_k$ for some j and k with $0 \leq j \leq k < i$. Therefore, if U is an addition chain that computes e as mentioned above, then for any $\alpha \in [1, n-1]$ we can find $\beta = \alpha^e \bmod n$ by successively computing: $\alpha, \alpha^{u_1}, \ldots, \alpha^{u_{l-1}}, \alpha^e$.

Let $l(e)$ be the shortest length of any valid addition chain for a given positive integer e. Then the theoretical minimum number of field multiplications required for computing the modular exponentiation of (1) is precisely $l(e)$. Unfortunately, the problem of determining an addition chain for e with the shortest length $l(e)$ is an **NP**-hard problem [2].

Across the centuries, a vast amount of algorithms for computing modular exponentiation have been reported. Reported strategies include: binary, m-ary, adaptive m-ary, power tree, the factor method, etc. [3, 1, 2]. On the other hand, relatively few probabilistic heuristics have been reported so far for finding near optimal addition chains [4, 5]. In this paper, we present a Genetic Algorithm (GA) suited to optimize addition chains. The results obtained suggest that this approach is a very competitive alternative to the solution of the problem.

2 Problem Statement

The problem addressed in this work consists of finding the shortest addition chain for an exponent e. Formally, an addition chain can be defined as follows,

Definition. An *addition chain* U for a positive integer e of length l is a sequence of positive integers $U = \{u_0, u_1, \cdots, u_l\}$, and an associated sequence of r pairs $V = \{v_1, v_2 \cdots, v_l\}$ with $v_i = (i_1, i_2), 0 \leq i_2 \leq i_1 < i$, such that:

$$u_0 = 1 \text{ and } u_l = e; \quad \text{for each } u_i, 1 \leq i \leq l, u_i = u_{i_1} + u_{i_2}.$$

The search space for computing optimal addition chains increments its size at a factorial rate as there exist $r!$ different and valid addition chains with length r. Clearly, the problem of finding the shortest ones becomes more and more complicated as r grows larger.

3 Deterministic Heuristics for Modular Exponentiation

In this section, we briefly review some deterministic heuristics proposed in the literature for computing field exponentiation. For a complete description of these and other methods, interested readers are referred to [1, 6].

Let e be an arbitrary m-bit positive integer e, with a binary expansion representation given as, $e = (1e_{m-2} \dots e_1 e_0)_2 = 2^{m-1} + \sum_{i=0}^{m-2} 2^i e_i$. Then,

$$y = x^e = x^{\sum_{i=0}^{m-1} 2^i e_i} = \prod_{i=0}^{m-1} x^{2^i e_i} = \prod_{e_i \neq 0} x^{2^i} \tag{2}$$

Binary strategies evaluate equation (2) by scanning the bits of the exponent e one by one, either from left to right (MSB-first binary algorithm) or from right to left (LSB-first binary algorithm) applying Horner's rule. Both strategies require a total of $m-1$ iterations. At each iteration a squaring operation is performed, and if the value of the scanned bit is one, a subsequent field multiplication is performed. Therefore, the binary strategy requires a total of $m-1$ squarings and $H(e)-1$ field multiplications, where $H(e)$ is the Hamming weight of the binary representation of e. The binary method can be generalized by scanning more than one bit at a time. Hence, the window method (first described in [1]) scans k bits at a time. The window method is based on a k-ary expansion of the exponent, where the bits of the exponent e are divided into k-bit words or digits. The resulting words of e are then scanned performing k consecutive squarings and a subsequent multiplication as needed. For $k = 1, 2, 3, 4$ the window method is called, respectively, *binary*, *quaternary octary* and *hexa* MSB-first exponentiation method.

4 The Proposed Genetic Algorithm

In this work, we present a Genetic Algorithm (GA) approach suited for finding optimal addition chains. In the rest of this Section we describe how the design decisions were taken.

4.1 Representation

In this work, we adopt an integer encoding, using variable-length chromosomes. Each element from the addition chain is directly mapped on each gene in the chromosome. Then, in this case, the genotype and the phenotype are both the same.

For example, if we are minimizing the addition chain for the exponent $e = 6271$, one candidate solution could be $1-2-4-8-10-20-30-60-90-180-360-720-1440-2880-5760-5970-6150-6240-6270-6271$.

This integer sequence represents a chromosome or individual I, where each gene I_k corresponds to one step on the addition chain, for $0 \leq k \leq l$ with length $l = 19$, and $I_l = 6271$.

4.2 Fitness Function

Since we are looking for the minimal addition chain's length, then the individual's fitness is precisely the addition chain's length, or in other words, the

chromosome's length. The shorter the chromosome's length is, the better its fitness value, and vice versa.

If we consider the previous addition chain, the associated fitness of this chain is 19.

4.3 Crossover Operator

The crossover operator creates two children from two parents. In our GA, we adopt *one point crossover*. Some extra considerations must be taken, however, mainly because of two reasons: first, it is necessary to assure that the resulting children are valid addition chains (i.e., feasible ones) and second, the chromosomes are of variable length. The way this operator produces offspring is illustrated in Figure 1. The genes before the crossover point are copied to the children as values (the alleles are copied); meanwhile from the genes after the crossover point only the rules are copied (rules indicate the positions from the chains which are selected to be added). See the following pseudocode:

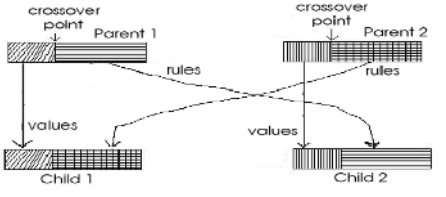

Fig. 1. Crossover Operator

Begin Function Crossover
For a pair of parents ($P1$ and $P2$) do:

1. Select a randomly selected crossover point p such that $(2 \leq p \leq l-2)$ where l is the chromosome length.
2. Create child ($C1$), copy the P1's values to the C1 starting from 0 until p is reached, hence,
   For $(k = 0)$ to $(k = p)$ $C1_k \leftarrow P1_k$
   From the point p until e is reached, complete the child C1 following the rules by which P2 was created, in the following way,
   For $(k = p + 1)$ to $(k = length)$
   - Look for a and b values such that, $P2_k = P2_a + P2_b$
   - Set $C1_k \leftarrow C1_a + C1_b$

   EndFor
3. Create child ($C2$):
   For $(k = 0)$ to $(k = p)$
   - $C2_k \leftarrow P2_k$

   EndFor
   For $(k = p + 1)$ to $(k = length)$
   - Look for a and b values such that, $P1_k = P1_a + P1_b$
   - Set $C2_k \leftarrow C2_a + C2_b$

   EndFor

End Function Crossover
We point out that the crossover operator is applied in the way described above only when that data manipulation does not produce values exceeding the exponent e. In case the value indicated by the crossover operator exceeds e, then the value assigned would be the maximum allowable.

4.4 Mutation Operator

It is noted that our definition of this operator allows us to introduce random changes into the chromosome, while preserving addition chains' validity.
BEGIN Function Mutation
For each child (C) do:

1. Randomly select a mutation point i and a random number j such that $2 \leq j < i < (l-2)$, where l is the chromosome length.
2. The new value of the child at the mutation point C_{i+1} will be $C_{i+1} = C_i + C_j$
3. Repair the upper part of the chromosome $\{C_{k>i+1}\}$, using the following criterion:
   For $k = i+2$ to l, with $C_l = e$ do
   If (Flip(Z)) then use the doubling rule whenever is possible, i.e, $C_k = 2C_{k-1}$
   Else *if (flip(0.5))* set $C_k = C_{k-1} + C_{k-2}$
   Else set $C_k = C_m + C_n$, where m and n are two randomly selected integers such that $0 \leq m, n < l$.

END Function Mutation

$Flip(Z)$ is a function that receives an input parameter Z such that $0 \leq Z \leq 1$. It returns $true$ with probability Z, or $false$ in other case.

General GA
Having defined the main Genetic Algorithm primitives, we proceed to put them together into the skeleton structure of the GA strategy outlined below,

BEGIN-General-GA
1.Randomly create an initial population size N.
2.Repeat:
   2.a. Compute individuals' fitness.
   2.b Select the N parents to be reproduced.
   2.c With probability Pc, apply crossover operator to the N parents.
   2.d Apply the mutation operator to the children with a probability Pm.
   2.e. Children will form the next generation population.
3. Go to step 2.a. until a predetermined number of *Generations* is reached.
4. Report the fittest individual.
END-General-GA.

5 Experiments and Results

In order to validate the GA approach described in the previous Section, we conducted a series of experiments, with the aim of comparing our GA's experimental results against the ones obtained by using several traditional deterministic methods.

Table 1. Accumulated addition chain lengths for exponents $e \in [1, 1000]$

Optimal value=**10808**			
Strategy	**Total length**	**Strategy**	**Total length**
Dyadic [6]	10837	Quaternary	11479
Total [6]	10821	Genetic Algorithm	Best: 10818
Fermat [6]	10927		Average: 10824.07
Dichotomic [6]	11064		Median: 10824
Factor [6]	11088		Worst: 10830
Binary	11925		Std.Dev.: 2.59

The first set of experiments consisted on finding the accumulated addition chain lengths for all exponents e in a given interval as it was done in [6]. Then, as a second test, we applied our genetic algorithm to a special class of exponents whose optimal addition chains are particularly hard to find. All our experiments were performed by applying the following GA's parameters: Population size $N = 100$, Number of Generations = 300, Crossover Rate $Pc = 0.6$, Mutation Rate $Pm = 0.5$, Probability $Z = 0.7$ (used in the mutation operator), Selection = Binary Tournament. All the statistical results shown here were produced from 30 independent runs of the algorithm with different and independent random seeds (adopting a uniform distribution).

Using our genetic algorithm approach, we computed the accumulated addition chain lengths for all the first 1000 exponents, i.e. $e \in [1, 1000]$. The accumulated value so obtained was then compared against the accumulated values reported in [6] by applying the following deterministic methods: Dyadic, Total, Fermat, Dichotomic, Factor, Quaternary and Binary [1, 6]. All results found are shown in Table 1. It can be seen that in the best case, our GA approach obtained better results than all the other six methods. In average, the GA approach was ranked in second place, only behind the Total method. It is noted that none of the features strategies was able to find the optimal value that was found by performing an exhaustive search.

Furthermore, we computed the accumulated addition chain lengths for all the exponents in the ranges $e \in [1, 512]$, $e \in [1, 2000]$ and $e \in [1, 4096]$. The methods used were the GA strategy, the binary method and the quaternary method. Once again and for comparative purposes, we computed the corresponding optimal values (obtained by enumeration). Those results are shown in Table 2.

Clearly, the results obtained by the GA strategy outperformed both, the binary and the quaternary method, even in the worst case. We can observe that for the three cases considered (i.e., 512, 2000 and 4096), the GA obtained a reasonably good approximation of the optimal value.

A Special Class of Exponents Hard to Optimize

Let $e = c(r)$ be the smallest exponent that can be reached using an addition chain of length r. Solutions for that class of exponents are known up to $r = 30$ and a compilation of them can be found in [7]. Interesting enough, the computational difficulty of finding shortest addition chains for those exponents seems to be among the hardest if not the hardest one from the studied exponent families [1]. We show the solutions found by the GA for the class of exponents shown in Table 3.

Table 2. Accumulated addition chain lengths for 512, 2000 and 4096

for all $e \in [1, 512]$	for all $e \in [1, 2000]$	for all $e \in [1, 4096]$
Optimal: 4924	Optimal: 24063	Optimal: 54425
Binary: 5388	Binary: 26834	Binary: 61455
Quaternary: 5226	Quaternary: 25923	Quaternary: 58678
Genetic Algorithm Best: 4925 Average: 4927.7 Median: 4927 Worst: 4952 Std.Dev.: 4.74	Genetic Algorithm Best: 24124 Average: 24135.17 Median: 24136 Worst: 24144 Std.Dev: 5.65	Genetic Algorithm Best: 54648 Average: 54684.13 Median: 54685 Worst: 54709 Std.Dev.: 13.55

Table 3. Shortest addition chains for a special class of exponents

exponent $e = c(r)$	Addition Chain	Length r
1	1	0
2	$1-2$	1
3	$1-2-3$	2
5	$1-2-3-5$	3
7	$1-2-3-5-7$	4
11	$1-2-3-5-8-11$	5
19	$1-2-3-5-7-12-19$	6
29	$1-2-3-4-7-11-18-29$	7
47	$1-2-3-5-10-20-40-45-47$	8
71	$1-2-3-5-7-12-17-34-68-71$	9
127	$1-2-4-6-12-24-48-72-120-126-127$	10
191	$1-2-3-5-10-20-21-42-63-126-189-191$	11
379	$1-2-4-5-10-15-25-50-75-150-300-375-379$	12
607	$1-2-4-6-12-24-48-96-192-384-576-600-606-607$	13
1087	$1-2-3-6-12-18-36-74-144-216-432-864-865-1081-1087$	14
1903	$1-2-3-5-10-13-26-52-104-105-210-420-840-1680-1890$ 1903	15
3583	$1-2-3-6-12-18-36-72-108-216-432-864-1728-3456$ $-3564-3582-3583$	16
6271	$1-2-3-6-12-24-48-96-192-384-768-1536-3072-6144$ $-6240-6264-6270-6271$	17
11231	$1-2-3-6-12-24-25-50-100-200-400-800-1600-3200$ $-6400-9600-11200-11225-11231$	18
18287	$1-2-3-6-9-15-30-45-47-94-188-190-380-760-1520$ $-3040-6080-12160-18240-18287$	19
34303	$1-2-3-6-12-14-28-56-112-224-448-504-1008-2016$ $-4032-8064-16128-32256-34272-34300-34303$	20
65131	$1-2-3-6-12-24-48-72-144-288-576-1152-2304-4608$ $-4611-9222-18444-27666-55332-55908-65130-65131$	21
110591	$-1-2-4-5-10-20-40-80-160-320-640-1280-2560-2570$ $5140-7710-12850-25700-51400-102800-110510-110590-110591$	22
196591	$1-2-3-6-12-15-30-60-120-240-480-720-1440-2880-5760$ $11520-23040-46080-92160-184320-19584-196560-196590-196591$	23
357887	$1-2-3-4-8-16-32-64-128-256-257-514-771-11542$ $-3084-6168-12336-24672-49344-49347-98691-148038-296076$ $-345423-357759-357887$	24
685951	$1-2-4-6-7-14-21-42-84-168-336-504-840-1680$ $-3360-6720-13440-26880-53760-57120-114240-228480$ $-342720-685440-685944-685951$	25
1176431	$1-2-4-5-10-15-19-38-76-152-304-608-612$ $-1224-2448-4896-9792-19584-29376-58752-117504-235008$ $-352512-587520-1175040-1176264-1176416-1176431$	27
2211837	$1-2-3-6-9-15-30-60-120-126-252-504-1008$ $-2016-4032-8062-16128-16143-32286-64572-129144-258288$ $-516576-1033152-2066304-2195448-2211591-2211717-2211837$	28
4169527	$1-2-3-6-12-24-48-96-192-384-768-1536-2304$ $-4608-9216-18432-36864-73728-147456-294912-589824-589825$ $-1179650-1769475-3538950-4128775-4165639$ $-414167943-4169479-4169527$	29
7624319	$1-2-3-6-12-18-36-72-144-288-576-1152-1224$ $-2448-4896-9792-19584-39168-78336-156672-313344$ $-626688-1253376-1254600-1274184-1274185-2548370$ $-5096740-6370925-7624301-7624319$	30
14143037	$1-2-3-6-12-18-30-60-120-240-480-960-961$ $-1922-3844-7688-11532-23064-46128-92256-184512-369024$ $-461280-830304-1660608-3321216-6642432$ $-13284864-14115168-14138232-14142076-14143037$	31

It is noted that in 24 out of 30 exponents, the GA approach was able to find the shortest addition chain. However, for the 6 remaining exponents (namely, 357887, 1176431, 2211837,4169527, 7624319 and 14143037), our GA strategy found addition chains that where one unit above the optimal.

6 Conclusions and Future Work

In this paper we described how a genetic algorithm strategy can be applied to the problem of finding shortest addition chains for optimal field exponentiation computations. The GA heuristic presented in this work was capable of finding almost all the optimal addition chains for any given fixed exponent e with $e < 4096$. Taking into account the optimal value (which was found by enumeration) the percentage error of our GA strategy was within 0.4% from the optimal for all cases considered. In other words, for any given fixed exponent e with $e < 4096$, our strategy was able to find the requested shortest addition chain in at least 99.6% of the cases. In a second experiment for assessing the actual power of the GA strategy as a search engine, we tested it for generating the shortest addition chains of a class of exponents particularly hard to optimize, whose optimal lengths happen to be known for the first 30 members of the family. In most cases considered, the GA strategy was able to find the optimal values.

Acknowledgments

The first and second authors acknowledge support from CONACyT through the CONACyT project number 45306. The third and fourth authors acknowledge support from CONACyT through the NSF-CONACyT project number 42435-Y.

References

1. Knuth, D.: The Art of Computer Programming. 3rd. edn. Addison-Wesley, Reading, Massachusetts (1997)
2. Menezes, A. J., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton, Florida (1996)
3. Gordon, D. M.: A Survey of Fast Exponentiation Methods. Journal of Algorithms, Vol. 27 No.1 (1998) 129–146
4. Cruz-Cortés, N., Rodríguez-Henríquez, F., Coello, C.: On the Optimal Computation of Finite Field Exponentiation. In: Lemaître, C., Reyes, C. and González, J. (eds.): Advances in Artificial Intelligence - IBERAMIA 2004: 9th Ibero-American Conference on AI. Lecture Notes in Computer Science, Vol. 3315. Springer-Verlag, (2004) 747–756
5. Bos, J., Coster, M.: Addition Chain Heuristics. In: Brassard, G. (ed.): Advances in Cryptology —CRYPTO 89. Lecture Notes in Computer Science, Vol. 435. Springer-Verlag, (1989) 400–407
6. Bergeron, F., Berstel, J., Brlek, S.: Efficient Computation of Addition Chains. Journal de thorie des nombres de Bordeaux, Vol. 6 (1994) 21–38
7. Bleinchenbacher, D., Flammenkamp, A.: An Efficient Algorithm for Computing Shortest Addition Chains. Available at: `http://www.uni-bielefeld.de/~achim` (1997)