Indian Statistical Institute Kolkata

M.Tech (Computer Science) Dissertation

Modifications of Bos and Coster's Heuristics in search of a shorter addition chain for faster exponentiation

A dissertation submitted in partial fulfillment of the requirements for the award of M.Tech.(Computer Science) degree

Author:
Ayan Nandy
Roll No:MTC0916

Supervisor:
Prof. Palash Sarkar
Applied Statistics Unit

Acknowledgements

At the end of this course, it is my pleasure to thank everyone who has helped me along the way.

First of all, I want to express my sincere gratitude to my supervisor, Prof. Palash Sarkar for giving me interesting problems. It was a memorable learning experience. For his patience, for all his advice and encouragement and for the way he helped me to think about problems with a broader perspective, I will always be grateful.

I would like to thank all the professors at the Indian Statistical Institute Kolkata who have made my educational life exciting and helped me to gain a better outlook on computer science. I would like to express my gratitude to Prof. K.C. Gupta, Somindu, Shashank and Sanjay for interesting discussions. Without the precious suggestions from Somindu, it would have been considerably difficult to finish this report.

I would like to thank everybody at Indian Statistical Institute Kolkata for providing a wonderful atmosphere for pursuing my studies. I thank all my classmates, seniors and juniors who have made the academic and non-academic experience very delightful. I would specially thank Shyam for his encouragement and valuable support before the semester examinations, my seniors Saurabh and Kaushik, classmates Simanta and Nibedita for their priceless support in helping me with the assignments and Sourav, Somabrata and my junior Amit for all the fun we could fit in between the days of hard work.

My most important acknowledgement goes to my family and friends who have filled my life with happiness. My deepest gratitudes are with my parents who have always encouraged me to pursue my passions and supported me in all weathers withstanding all their limitations and keeping aside all their priorities all along their lives. I also thank my sister Arpita, brother Arijit and friends Bunty, Kutu, Taniya, Biltu and Deba for their unending support, care and concerns.

Contents

Chapter 1

Introduction

The basic question that necessitates a study of addition chains is: What is the fewest number of multiplications necessary to compute g^r, given that the only operation permitted is multiplying two already-computed powers? This is equivalent to the question: What is the length of the shortest addition chain for r?

An addition chain for r is a list of positive integers $a_1 = 1, a_2, \ldots, a_\ell = r$, such that, for each $i > 1$, there is some j and k with $1 \leq j \leq k < i$ and $a_i = a_j + a_k$.

A short addition chain for r gives a fast algorithm for computing g^r : compute $g^{a_1}, g^{a_2}, \ldots, g^{a_{\ell-1}}, g^r$. Let $\ell(r)$ be the length of the shortest addition chain for r. The exact value of $\ell(r)$ is known only for relatively small values of r. It is known that, for large r, $\quad \ell(r) = \log r + \frac{(1+o(1))\log r}{\log\log r}$

The lower bound was shown in [Erdos60] using a counting argument and the upper bound is given by the m-ary method discussed in 2.1. In 2, few basic methods and the double base number system used for fast exponentiation are discussed. In 3, outline of few algorithms for finding an addition chain containing a number n has been provided. 4 constitutes the three different algorithms we have designed as modifications to the Makesequence algorithm of Bos and Coster [BosCoster90]. In 5, in the first part we discuss the results obtained from running the C programs for Brauer's and Yao's algorithm for various n and k. In the later part we compare our algorithms with the original Bos and Coster method with respect to a test example.

Chapter 2

Basic Methods

2.1 Binary Method

This method is also known as the "square and multiply" method. The basic idea is to compute g^r using the binary expansion of r.

Let $r = \Sigma_{i=0}^{\ell} c_i 2^i$

Then the following algorithm will compute g^r :

$a \leftarrow 1$

for $d = \ell$ to 0 by -1

$a \leftarrow a * a$

if $c_d = 1$ then $a \leftarrow a * g$

return a.

At each step of the *for* loop, a is equal to g^s, where the binary representation of s is a prefix of the binary representation of r. Squaring a has the effect of doubling s, and multiplying by g puts a 1 in the last digit, if the corresponding bit c_i is 1. [Knuth81] gives a right-to-left version of the algorithm, which has the advantage of not needing to know ℓ ahead of time.

This algorithm takes $2\lfloor \log r \rfloor$ multiplies in worst case, and $\frac{3\lfloor \log r \rfloor}{2}$ on average. Since $\lfloor \log r \rfloor$ is a lower bound for the number of multiplies needed to do a single exponentiation in a general group, this method is often good enough.

2.2 m-ary method

Let $r = \Sigma_{i=0}^{\ell} c_i m^i$

The m-ary method computes g^r using this representation:

Compute $g^2, g^3, \ldots, g^{m-1}$

$a \leftarrow 1$

for $d = \ell$ to 0 by -1

$a \leftarrow a^m$

$a \leftarrow a g^{c_d}$

return a

2.3 Window Method

The 2^k-ary method may be thought of as taking k-bit windows in the binary representation of r, calculating the powers in the windows one by one, squaring them k times to shift them over, and then multiplying by the power in the next window. This leads to several different generalizations. One obvious generalization is that there is no reason to force the windows to be next to each other. Strings of zeros do not need to be calculated, and may be skipped. Moreover, only odd powers of g need to be computed in the first step.

For example the binary representation of $r = 26235947428953663183191$ is given by

1011000111001000000111010010100111010100000010111100000111110011

00101010111.

[Gordon98] showed that the optimal choice for the m-ary method for this 75-bit number is $m = 8$, which takes 102 multiplications. For the window method, with windows of length up to 4, the number of multiplies is only 93; 8 multiplies to compute the odd powers up to 15, 71 squaring, and 14 multiplies for the intermediate values:

$$\underbrace{1011}_{11}000\underbrace{111}_{7}00\underbrace{1}_{1}000000\underbrace{111}_{7}0\underbrace{1001}_{9}0\underbrace{1001}_{9}\underbrace{1101}_{13}0\underbrace{1}_{1}000000\underbrace{1011}_{11}\underbrace{11}_{3}$$

$$00000\underbrace{1111}_{15}\underbrace{1001}_{9}\underbrace{1001}_{9}0\underbrace{101}_{5}0\underbrace{111}_{7}$$

[BosCoster90] suggested using larger windows. Instead of constructing a table of all odd numbers less than m, they use an addition sequence to compute all the intermediate values needed for this particular exponentiation. An addition sequence is essentially a sequence of integers consisting of all the window elements (47, 117, 343, 499, 933, 5689 in the following example) such that it is an addition chain for the

largest window element (5689 in this example). Using large windows can reduce the number of multiplies to 89:

$$\underbrace{1011000111001}_{5689}000000\underbrace{1110100101}_{933}00\underbrace{1110101}_{117}000000\underbrace{101111}_{47}00000\underbrace{111110011}_{499}$$

$$00\underbrace{101010111}_{343}$$

They use 62 squaring, 5 multiplications of intermediate values, and 22 multiplications to compute the addition sequence 1; 2; 4; 8; 10; 11; 18; 36; 47; 55; 91; 109; 117; 226; 343; 434; 489; 499; 933;1422; 2844; 5688; 5689.

The Window method is the first part of the Bos and Coster's algorithm that reduces the computation of an addition chain for n to the computation of an addition sequence that contains a given set of numbers which are much smaller than n. An overview of the second part producing the addition sequence consisting of the Window elements is given in 3.3.

2.4 Double-Base Number System

The Double-Base Number System (DBNS) was initially introduced by Dimitrov and Cooklev [Dimitrov95] and later used in the context of elliptic curve cryptography [Imbert2005]. With this system, an integer k is represented as

$$k = \Sigma_{i=1}^{\ell} c_i 2^{a_i} 3^{b_i}$$

with $c_i \in \{1, -1\}$

To find an expansion representing k, we can use a greedy-type algorithm whose principle is to find at each step the best approximation of a certain integer k initially in terms of a 2,3-integer, i.e. an integer of the form $2^a 3^b$ from a suitable table as explained by Doche et al [Doche2009] which we refer here and in 4.1.2 as DKS table. Then the difference is computed and the process is reapplied.

Algorithm for a double-base chain computing k

Input: $k \geq 0$

Output: $k = \Sigma_{i=1}^{\ell} s_i 2^{a_i} 3^{b_i}$ with (a_i,b_i) decreasing.

while $k \neq 0$ do
    $s = 1$
    Find the best default approximation of k of the form $z = 2^a 3^b$ with $a \leq a_{\max}$
    and $b \leq b_{\max}$ from DKS table

Print(s,a,b)
$a_{\max} = a; b_{\max} = b$
if $k < z$ then $s = -s$
$k = |k - z|$
end while

Example: Applying this approach for $n = 542788$, we find that

$542788 = 2^8 3^7 - 2^3 3^7 + 2^4 3^3 - 2.3^2 – 2$

In [Imbert2005], Dimitrov et al. show that for any integer n, this greedy approach returns a DBNS expansion of n having at most $O(\frac{\log n}{\log \log n})$ terms. However, in general this system is not well suited for scalar multiplications. Indeed, if it is impossible to order the terms in the expansion such that their powers of 2 and 3 are simultaneously decreasing, as it is the case in Example 1, it seems impossible to obtain $[n]P$ without using too many doublings or triplings and without extra temporary variables.

This observation leads to the concept of Double-Base Chain (DBC), introduced in [Imbert2005], where Dimitrov et al. explicitly look for expansions such that $a_\ell \geq a_{\ell-1} \geq \ldots \geq a_1$ and $b_\ell \geq b_{\ell-1} \geq \ldots \geq b_1$. This guarantees that exactly a_ℓ doublings and b_ℓ triplings are needed to compute $[n]P$. it is easy to modify the greedy algorithm to return a DBC.

Chapter 3

Some algorithms for finding an addition chain

3.1 Brauer's Algorithm

Brauer's chain $B_k(n)$ is an addition chain containing n which is parametrized by a positive number k and defined recursively as follows:

$$B_k(n) = \begin{cases} 1, 2, \ldots, 2^k - 1 & n < 2^k \\ B_k(q), 2q, 4q, 8q, \ldots, 2^k q, n & n \geq 2^k, q = \lfloor \frac{n}{2^k} \rfloor \end{cases}$$

To eliminate repetition, avoid even numbers (other than 2) less than 2^k and allow flexibility in the exponents to reduce the length of the chain, we get the following popular chain defined recursively for $k \geq 2$ and $n \geq 2^k$ as follows:

$$T_k(n) = \begin{cases} 1, 2, 3, 5, 7, 9, \ldots, 2^k - 1, n & \text{if } n < 2^{k+1} \text{ and } n \text{ is even} \\ T_k(n/2), n & \text{if } n \geq 2^{k+1} \text{ and } n \text{ is even} \\ T_k(n - (n \bmod 2^{\lceil \log n \rceil - k})), n & \text{if } n < 2^{2k} \text{ and } n \text{ is odd} \\ T_k(n - (n \bmod 2^k)), n & \text{if } n \geq 2^{2k} \text{ and } n \text{ is odd} \end{cases}$$

Strauss generalized Brauer's algorithm to compute a product $x_1^{n_1} x_2^{n_2} \ldots x_p^{n_p}$, i.e., to obtain the vector (n_1, n_2,..., n_p) by additions starting from the unit vectors. Strauss's algorithm writes (n_1, n_2,..., n_p) in radix 2^k, where each coefficient is a vector (r_1, r_2,..., r_p) with r_1, r_2,..., $r_p \in \{1, 2, \ldots, 2^k - 1\}$

3.2 Yao's Algorithm

Yao's Algorithm finds an addition chain containing n where the chain is parametrized by a positive number k. Initially n is written in radix 2^k as $\Sigma_{i=0}^{j} c_i 2^{ik}$ with $c_j \neq 0$.

Define $d(z)$ as the sum of 2^{ik} over all i such that $c_i = z$.

Yao's chain begins with 1,2,4,8,...,$2^{\lfloor \log n \rfloor}$; adds various 2^{ik} to obtain $d(z)$ for each $z \in \{0, 1, \ldots, 2^k - 1\}$ such that $d(z)$ is non-zero; then obtains $zd(z)$ for each z and finally obtains

$$n = d(1) + 2d(2) + 3d(3) + ... + (2^k - 1)d(2^k - 1)$$

3.3 The Makesequence algorithm by Bos and Coster

Bos and Coster describe a routine that makes an addition sequence of a set of numbers. It starts with a Protosequence consisting of 1,2 and the requested numbers, i.e., the Window elements sent by the Window Method explained in 2.3. It then transforms this to another one using a heuristic algorithm. In each step, the Protosequence is reduced to a simpler one, having smaller numbers.

A Protosequence is written as 1, 2,...,f_2, f_1,f. The prefix *proto* means first and hence the Protosequence is the initial sequence delivered by the Window Method which is not necessarily an addition sequence, i.e, there might be a Window element which can not be expressed as the sum of two other Window elements.. Initially the Makesequence is a copy of the Protosequence. The algorithm inserts some numbers in the Makesequence by one of the four methods described below and at each step a Window element f leaves the Protosequence. When only 1 and 2 are left in the Protosequence, the Makesequence takes the shape of an addition chain containing of all the Window elements. Since the Makesequence delivers all the Window elements, a simple double and add scheme will lead us to an addition chain for the large integer we started with using the Window Method. The Makesequence algorithm is less costlier than the 2^k-ary method since here we need not have a lookup table with the $2^{k-1} - 1$ exponents.

3.3.1 Approximation

There are two elements a and b in the sequence with $a + b = f - \epsilon$, where ϵ is positive and small; insert $a + \epsilon$ in the Makesequence. Since $a + \epsilon$ and b add up to f, we delete f from the Protosequence.

Condition: $0 \leq f - (f_i + f_j) = \epsilon; f_i \leq f_j$ where $i, j \in \mathbf{N}$.

Insert: $f_i + \epsilon$

Example: 49 67 85 117
ϵ =1 [117-(49+67)]

Insert: 50

Result: 49 <u>50</u> 67 85 (117)

3.3.2 Division

f is divisible by a small prime p; put $\frac{f}{p}, \frac{2f}{p}, \ldots, f$ in the sequence and delete f from the Protosequence. This operation is equivalent to finding an addition chain 1,2,..., p, multiplying each element with $\frac{f}{p}$ and inserting the resulting numbers in the Makesequence.

Condition: p= 3,5,7 or 17. f is divisible by p.

Insert: $\frac{f}{p}, \frac{2f}{p}, \ldots$

Example: 17 48

Insert: 16,32

Result: <u>16</u> 17 <u>32</u> (48)

3.3.3 Halving

Take a small number s that occurs earlier in the sequence, and put $f - s, \frac{f-s}{2}, \frac{f-s}{4}, \ldots$ to a certain point in the sequence.

Condition: $\frac{f}{f_1} \geq 2^u; \lfloor \frac{f}{2^u} \rfloor = k$.

Insert: $d = f - k2^u, f - d = k2^u, k2^{u-1}, \ldots, 2k, k$.

Example: 14 382; $\frac{f}{f_1} = 27.2; u = 4; k = 23; d = 14$.

Insert: 23 46 92 184 368

Result: 14 <u>23</u> <u>46</u> <u>92</u> <u>184</u> <u>368</u> (382)

3.3.4 Lucas

A Lucas sequence is an integer sequence that satisfies the recurrence relation $u_{n+1} = u_n + u_{n-1}$. In the given sequence, put a Lucas sequence that has f as last element.

Condition: f and f_i are the elements of a Lucas series (i.e., $f_i = u_0$ and $f = u_k, \geq 3$)

Insert: $u_1, u_2, \ldots, u_{k-1}$.

Example: 4 23

Insert: 5 9 14

Result: 4 5 9 14 (23)

Bos and Coster [BosCoster90] didn't mention in his heuristics how to search for a Lucas sequence among the window elements or how to ensure that we are selcting the best Lucas sequence.

After applying one of those insertions, the process is repeated until the sequence contains only the numbers 1 and 2. Let P denote the set of all the numbers apart from 1,2 which were present in the Protosequence. Any number in P can be expressed as a sum of two numbers taken from $P \cup I$, where I is the set of inserted numbers. Then $\{1, 2\} \cup P \cup I$ consists of all the numbers of the required addition chain.

To summarize, given a positive integer n to calculate x^n for some x, we do the following:

1. Express n in binary representation
2. Use the Window method as explained in 2.3 to get a Protosequence
3. The Makesequence routine reduces the Protosequence at each step and inserts new numbers in the sequence being made until the Protosequence has only 1 and 2 in it.
4. We get an addition chain $a_1 = 1, a_2 = 2, \ldots, a_\ell = n$, such that, for all $i > 1$, there is some j and k with $1 \leq j \leq k < i$ and $a_i = a_j + a_k$.
5. Since for all $i > 1$, there is some j and k with $1 \leq j \leq k < i$ and $a_i = a_j + a_k$, for all $i > 1$, there is some j and k with $1 \leq j \leq k < i$ and $x^{a_i} = x^{a_j} x^{a_k}$.
6. In $\ell - 1$ steps x^n can be calculated from x.

3.4 Fast Exponentiation using Data Compression

The exponentiation algorithm of [Yacovi99] uses the entropy of the source of the exponent to improve on existing exponentiation algorithms when the entropy is smaller than $\frac{\ell(S)}{1+w(S)}$ where $w(S)$ is the Hamming weight of the exponent, and $\ell(S)$ is its length.

3.4.1 The Lempel Ziv theory of Data Compression

Let A^* denote the set of all finite-length sequences over a finite alphabet A. Let $\ell(S)$ denote the length of $S \in A^*$ and let $A^n = \{S \in A^* | \ell(S) = n\}, n \geq 0$. The null sequence χ is in A^* .

Let $S(i, j)$ denote a substring of S that starts at location i and ends at location j. Let $S\pi^i$ denote $S(1, \ell(S) - i), i = 0, 1, \ldots, \ell(S)$. The vocabulary, $v(S)$, of S is the subset of A^* formed by all the substrings

of S. When a sequence S is extended by concatenation with one of its words, say $W = S(i,j)$, the resulting sequence $R = SW$ can be viewed as being obtained from S through a copying procedure. The same recursive copying procedure could be applied to generate an extension $R = SQ$ of S which is much longer than warranted by any word in $v(S)$. The only provision is that Q be an element of $v(SQ\pi)$.

We use the denotation $S \Rightarrow R$ if $R = SQ$ can be obtained from S by application of the above copying procedure, where at the end of the copying process we use "one- symbol innovation" (any symbol from A, not subject to the copying procedure). This process is called reproduction, while a single-step copying without innovation is called production and is denoted $S \rightarrow R$. If R cannot be obtained from S by production we write $S \not\rightarrow R$.

For an m-step production process of a sequence S and let $S(1,h_i), i = 1,2,\ldots,m, h_1 = 1, h_m = \ell(S)$, be the m states of the process. The parsing of S into $H(S) = S(1,h_1)S(h_1+1,h_2),\ldots,S(h_{m-1}+1,h_m)$ is called the production history of S, and the m words $H_i(S) = S(h_{i-1}+1,h_i), i = 1,2,\ldots,m$, where $h_0 = 0$, are called the components of $H(S)$. A component $H_i(S)$ and the corresponding reproduction step $S(1,h_{i-1}) \Rightarrow S(1,h_i)$ are called exhaustive if $S(1,h_{i-1}) \not\rightarrow S(1,h_i)$; a history is called exhaustive if each of its components, with the possible exception of the last one, is such. Every nonnull sequence S has a unique exhaustive history (denoted $E(S)$).

Let $c_H(S)$ denote the number of components in a history $H(S)$ of S. The production complexity of S is defined as $c(S) = \min\{c_H(S)\}$, where minimization is over all histories of S. Let $c_E(S)$ denote the production complexity of the exhaustive history of S.

Lempel and Ziv [LempelZiv76] showed that

1. $\forall S, c(S) = c_E(S)$
2. $\forall S \in A^n, c(S) < \frac{n}{(1-\epsilon_n)\log n}$ where $\epsilon_n = 2\frac{1+\log\log \alpha n}{\log n}, \alpha = |A|$
3. for an ergodic α-symbol source with normalized entropy h $(0 \le h \le 1)$, $c(S) \le \frac{hn}{\log n}$

3.4.2 Using the LZ compression method for fast exponentiation

Given the exponent S, the computation of any exponentiation x^S proceeds as follows assuming that $S(1,1)$ is the least significant bit of S (S is used to denote both integer and sequence):

Build a binary tree where each path from the root to any node corresponds to some segment of the exponent $S(i,j)$, and the node contains the result of $x^{S(i,j)}$. The root contains $1 = x^0$ (0 corresponds to the string χ). Proceed inductively as follows. Suppose that the component $S(i,j)$ was already processed; i.e., the tree already contains a path from the root to some leaf which corresponds to this component, and the leaf contains the result of $x^{S(i,j)}$. Traverse the partial tree from the root according to the new (so far unscanned) bits of $S(j+1,\ldots)$ until a leaf is reached. Proceed with S having one more symbol. The new component contains now exactly one new untraversed symbol. Extend the tree according to the

new symbol, and mark the new branch with this symbol. Compute the value of the new leaf. This simple construction (without the exponentiation) is the heart of the LZ algorithm. Ziv and Lempel proved that this construction creates the exhaustive history of S, $H(S) = S(1, h_1)S(h_1 + 1, h_2) \dots S(h_{m-1} + 1, h_m)$, where each path from the root to any node corresponds to one of the components of $H(S)$, and hence the number of nodes in the tree is $c(S)$.

For depth i we need to do one squaring (to compute $X^{2^i} = (X^{2^{i-1}})^2$), and whenever the new bit is 1 we need to multiply that value by the value of the father. For random exponents we can expect that to happen in half the cases for a total cost of $\frac{c(S)}{2}$, and in general, for exponent of length $\ell(S)$ and expected Hamming weight $w(S)$, the expected number of cases where the new symbol is 1, is $c(S)\frac{w(S)}{\ell(S)}$. Thus the total expected complexity $C(S)$ is computed as a function of $\ell(S), c(S)$ and $w(S)$. Yacovi [Yacovi99] found the following asymptomatic upper bound

$$C(S) = \ell(S) + \sigma(S)h\frac{\ell(S)}{\log \ell(S)} + o(\frac{\ell(S)}{\log \ell(S)})$$

where $\sigma(S) = 1 + \frac{w(S)}{\ell(S)}$

Yacov improved on a straightforward LZ compression, by taking advantage of leading zeros. Leading zeros are not accounted for in the binary tree, thus reducing the tree size.

Chapter 4

Modifications of the Bos and Coster Algorithm

The algorithm for smaller addition chains by Bos and Coster [BosCoster90] consists of two parts, the Window Method and the Makesequence Algorithm. The Window Method is explained in 2.3, while the Makesequence Algorithm is explained in 3.3. In this Chapter, we describe the three different methods we have tried as modifications of the Bos and Coster Algorithm. In the first algorithm we modify the Makesequence part by employing the Double Base Number System as explained in 2.4. The second modification employs a method of one-thirding which has some similarities with the halving scheme of the Makesequence Algorithm. The third modification is based on expressing the exponent in ternary number system. Our algorithms build an addition chain where a number may not be a sum of two other numbers or double another number in the chain but it is a triple of some number. In elliptic curves over characteristic 3 fields, the point triple is fast in comparison to the point double where our algorithms will be useful.

4.1 Pure DBNS

In this section we assume that original schemes of the Bos and Coster's Makesequence Algorithm like Approximation, Division, Halving and Lucas or modification like one-thirding (as explained in 4.2) will not be employed and hence we termed it pure DBNS. Our algorithm consists of two parts, the Window method and the Makesequence. The window method is same as the one explained by Bos and Coster [BosCoster90], whereas the **modified** Makesequence employs the DKS table, as explained in 2.4 instead of Approximation, Division etc.

4.1.1 The Window Method

The Window method, as explained in 2.3, is utilized to form an addition chain for a large number where it identifies a set of windows (binary strings) and forwards the window elements to the next stage of Makesequence to make a sequence. Here the number is expressed in binary form and windows or sub-strings from this binary string are identified in such a way that all the non-zero elements of the string are covered by these windows. The advantage of this method lies in the fact that once we have a chain for these windows representing integers much smaller than the given target, simple double and add operations will lead us to our target.

1. Express the number n in binary form and define the binary string as s.

2. Define a parameter k and a set A. User can specify the value of k on the basis of the value of n, $A = \emptyset$.

3. while the length of s is greater than k
   Define sub = sub-string of k characters of s from the left
   Delete the zeroes, if any, in the right end of sub.
   $A = A \cup \{sub\}$.
   Redefine s by deleting from it sub and the zeroes that follow it.
   end while.

4. Delete the tail of zeroes from s.

5. $A = A \cup s$

An Illustrative example

Let $n = 26235947428953663183191$ and $k = 13$
The binary representation of n is

$$s = 10110001110010000001110101010011101010000001011110000011111001100101010111$$

.

The first window is the substring of s as identified below
1011000111001000000111010010100111010100000010111100000111110011001010101111

The first sub-string is $sub = \underline{1011000111001}$ which is inserted in A.
Therefore, A is updated to $\{1011000111001\}$ and s is truncated to
1110100101001110101000000101111000001111100110010101011 1.

After 5 iterations, $A = \{\underbrace{1011000111001}_{5689}, \underbrace{1110100101}_{933}, \underbrace{1110101}_{117}, \underbrace{101111}_{47}, \underbrace{111110011}_{499}\}$ and $s = \underbrace{101010111}_{343}$, i.e., the length of s is less than 13.

Hence, the final output of the Window Method is
$A = \{\underbrace{1011000111001}_{5689}, \underbrace{1110100101}_{933}, \underbrace{1110101}_{117}, \underbrace{101111}_{47}, \underbrace{111110011}_{499}, \underbrace{101010111}_{343}\}$

4.1.2 The Modified Makesequence

The Window Method delivers us a protosequence consisting of numbers which are the window elements. The protosequence is not necessarily an addition sequence, i.e, there can be elements which can not be expressed as sum of two other elements. A Makesequence, as the name suggests, bridges the gap by

inserting new numbers to **make** the **sequence**. While Bos and Coster [BosCoster90] used methods like Approximation, Division etc. to make the sequence, here we employ the DKS table to get a DBNS representation of the numbers in the sequence we receive after working upon the Window Method and then make the sequence by inserting in it the required numbers. For illustrating the algorithm, we use the same example introduced in 4.1.1.

1. Arrange all the elements of A in ascending order. Add 1 and 2 to the sequence and get a Protosequence P in which we need to introduce more numbers to get the required addition chain.
2. Define $P = 1, 2, f_m, f_{m-1}, \ldots, f_2, f_1, f_0$, $R = \emptyset, P' = \emptyset, i = 0$
3. while $i < m$
   $g' = 0$
   $g = f_i$
   $h = 1$
   While $g > 0$
   Find the best approximation of g from DKS table of the
   form $z = 2^a 3^b$ with $a \geq 0$ and $b \geq 0$
   Insert z in R
   $g' = g' + hz$
   if $g < z$, then $h = -h$
   $g = |g - z|$, insert g in P'
   end while
   $i = i + 1$
   end while
4. Mark all elements of R in DKS table
5. Find the shortest set of Manhattan paths in DKS table covering all the elements in R
6. Insert all the elements covered by these paths in P
7. Insert all elements from P' in P
8. Output P

An Illustrative example

The Window Method delivers us $A = 47, 117, 343, 499, 933, 5689$
Therefore, $P = 1, 2, 47, 117, 343, 499, 933, 5689$
Corresponding to $f_0 = 5689$, we have $z = 2^3 3^6 = 5832$

Since $g = 5689 < z = 5832$, we have $h = -1$

The new value of g is 5832 - 5689 = 143

For $i = 0$, i.e., f_0, we get $R = \{1, 144, 5832\}$ since 5689 = 5832 -144 + 1

For $i = 6$, we have $R = \{1, 3, 16, 27, 36, 48, 64, 144, 324, 432, 972, 5832\}$ since
5689 = 5832 - 144 + 1
933 = 972 -36 -3
8 117 = 144 - 27
47 = 48 -1
499 = 432 + 64 + 3
343 = 324 + 16 + 3

Table 4.1: DKS table with elements of R marked

1	2	4	8	16	32	64
3	6	12	24	48	96	
9	18	36	72	144	288	
27	54	108	216	432	864	
81	162	324	648	1296	2592	
243	486	972	1944	3888	7776	
729	1458	2916	5832	11664	23328	

Table 4.2: DKS table with elements on shortest Manhattan paths marked

1	**2**	**4**	**8**	**16**	**32**	**64**
3	6	**12**	24	**48**	96	
9	18	**36**	72	**144**	288	
27	54	**108**	216	**432**	864	
81	162	**324**	648	1296	2592	
243	486	**972**	1944	3888	7776	
729	1458	**2916**	**5832**	11664	23328	

Elements to be inserted in P in step 6 are 1, 2, 3, 4, 8, 9, 12, 16, 27, 32, 36, 48, 64, 108, 144, 324, 432, 972, 2916 and 5832

Elements to be inserted in P in step 7 are 143, 39, 67 and 19

The desired sequence for the running example is the output P = 1, 2, 3, 4, 8, 9, 12, 16, 19, 27, 32, 36, 39, 47, 48, 64, 67, 108, 117, 143, 144, 324, 343, 432, 499, 933, 972, 2916, 5689, 5832

4.2 One thirding

Bos and Coster proposed a Halving routine as explained in 3.3.3. Since we have in mind elliptic curves where the point triple is fast in comparison to the point double, we consider a one-thirding scheme, which is similar to Halving.

Take a small number s that occurs earlier in the sequence, and put $f - s$, $\frac{f-s}{3}$, $\frac{f-s}{9}$, ... to a certain point in the sequence.

Condition: $\frac{f}{f_1} \geq 3^u; \lfloor \frac{f}{3^u} \rfloor = k$.

Insert: $d = f - k3^u, f - d = k3^u, k3^{u-1}, \ldots, 3k, k$.

Example: 14 382; $\frac{f}{f_1} = 27.2; u = 3; k = 14; d = 4$.

Insert: 4 14 42 126 378

Result: 4 14 42 126 378 (382)

Here we consider three variants of the One-thirding algorithm. For each of them, the Window method is identical to the one employed in 4.1.

4.2.1 Case I: Express every element in terms of the smallest

The Window Method sends us a sequence of numbers A which does not necessarily from an addition chain. Define the Makesequence P to be A appended by 1 and 2, while the elements of A which can not be expressed as a sum of two distinct elements or a double or a triple of another element forms the Protosequence Q. Like any Makesequence algorithm, our objective is simultaneously delete one element from Q and insert few elements in P in each step until P is an addition chain.

The initial set of steps are to express every element of A in terms of the smallest element in it, i.e., for each f_i we get a k near the smallest element f_r, such that tripling k for u times gives us an approximation of f_i which is a small number d away from f_i. At each of these steps f_i is deleted from the Protosequence while k and d are inserted in Q if they can not be expressed as sum of two distinct numbers or as double or triple of another number in the Makesequence P. If at the end of these steps Q is empty, P is the desired final Makesequence, i.e., an addition chain in itself. If Q is non empty, all except it's smallest element are sent to P and the consecutive differences after having the original elements of Q arranged in ascending order are kept in Q unless they can be expressed as sum of two distinct numbers or as double or triple of another number in the Makesequence P. At the final step the Protosequence Q is empty and at the same time the Makesequence P is an addition chain.

The Modified Makesequence

1. Arrange all the elements of A in ascending order.

2. Add 1 and 2 to the sequence and get a Protosequence P in which we need to introduce more numbers to get the required addition chain.

3. Define $P = 1, 2, f_m, f_{m-1}, \ldots, f_2, f_1, f_0$, Q = elements in P which can not be expressed as sum of two smaller elements or a triple of any single smaller element, $i = 0$, r = Number of elements in Q - 1.

4. While $i < r$
   Update r and arrange elements of Q in descending order as
   $q_0, q_1, \ldots, q_r$.
   $u = \lfloor \log_3(\frac{q_i}{q_r}) \rfloor, u_1 = \lfloor \frac{q_i}{3^u} \rfloor$.
   For $u > 2$, insert $d = q_i - u_1 3^u, q_i - d = u_1 3^u, u_1 3^{u-1}, \ldots, 3u_1, u_1$ in
   P and delete all of them as well as q_i from Q.
   Insert d and u_1 in Q.
   $i = i + 1$
   end while

5. If $Q \neq \emptyset$, arrange Q in ascending order, take the consecutive differences.
6. if any element in Q is a sum of two or three elements in P, and in case of three-sum insert the missing sum of two to P.
7. Else send elements of Q except smallest one to P and send the consecutive differences to Q until $Q = \emptyset$
8. Output P

An Illustrative example

Q = 47, 117, 343, 499, 933, 5689
$r = 5$

$i = 0$, $q_i = 5689$, $q_r = 47$

$u = 4$, u_1 =70

$d = q_i - u_1 3^u = 19$, $q_i - d = u_1 3^u = 3^4.70 = 5670$
Insert 19, 5670, 1890, 630, 210, 70 in P, remover 5689 from Q and insert 19 and 70 in Q
P = 1,2, 19, 47, 70, 117, 210, 343, 499, 630, 933, 1890, 5670, 5689
Q = 19, 47, 70, 117, 343, 499, 933

P = 1,2, 19, 47, 70, 117, 210, 343, 499, 630, 933, 1890, 5670, 5689, 34, 102, 306, 918, 15, 45, 135, 405, 94
Q = 15, 19, 34, 47, 70, 94, 117, 343

Iteration 1: Q = 4, 13, 23, 24, 226
Iteration 2: 226 = 15 + 94 + 117, Insert 4, 13, 23, 24, 109 (15+94) to P, Q = 4, 9, 10
Iteration 3: 9 = 1+4 + 4, 10 = 1+ 9, Insert 8 (4+4), 9, 10 into P, $Q = \emptyset$

P = 1, 2, 4, 8, 9, 10, 13, 15, 19, 23, 24, 34, 45, 47, 70, 94, 102, 109, 117, 135, 210, 343, 405, 499, 630, 918, 933, 1890, 5670, 5689

4.2.2 Case II: Express larger elements in terms of the third largest element

Here the Makesequence has two stages. The Stage I is similar to Case I. While in Case I all the elements were being expressed in terms of the smallest element, in Stage I of Case II, the larger elements are expressed in terms of the third largest element.

In Stage II, we run the usual Bos and Coster heuristics to searching for missing elements in the chain.

Stage I:

1. Arrange all the elements of A in ascending order.
2. Add 1 and 2 to the sequence and get a Protosequence P in which we need to introduce more numbers to get the required addition chain.
3. Define $P = 1, 2, f_m, f_{m-1}, \ldots, f_2, f_1, f_0$, Q = elements in P which can not be expressed as sum of two smaller elements, $i = 0$, r = Number of elements in Q
4. Update r and arrange elements of Q in descending order as $q_0, q_1, ..., q_r$
5. while $i < 2$
   $u = \lfloor \log_3(\frac{q_i}{q_2}) \rfloor, u_1 = \lfloor q_i 3^u \rfloor$
   Insert $d = q_i - u_1 3^u, q_i - d = u_1 3^u, u_1 3^{u-1}, .., 3u_1, u_1$ in P
   Delete all of them as well as q_i from Q.
   Insert d and u_1 in Q.
   $i = i + 1$
   end while
6. If $Q \neq \emptyset$, arrange elements in Q in ascending order, take the consecutive differences
7. if any element in Q is a sum of two or three elements in P, and in case of three-sum insert the missing sum of two to P
8. $A = A \cup P$

Stage II:

1. Arrange all the elements of A in ascending order. Add 1 and 2 to the sequence and get a Protosequence P in which we need to introduce more numbers to get the required addition chain.
2. Define $P = 1, 2, f_m, f_{m-1}, \ldots, f_2, f_1, f_0$, Q = elements in P which can not be expressed as sum of two smaller elements $P_0 = \emptyset$, $s_A = 0$, $s_D = 0$, $s_H = 0$, $s_L = 0$, N_A, N_D, N_H, N_L
3. while $Q \neq \emptyset$

4. **Lucas**

while $s_L \leq N_L$
Arrange elements of Q in descending order as $q_0, q_1, \ldots, q_r$
for j=1 to r
for i=0 to $j-1$
If $q_i - q_j$ is in P_0, delete q_i from Q
If $j < r$, $j = j+1$, $s_L = s_L + 1$
If $i < r-1$, $i = i+1$, $s_L = s_L + 1$
end while

5. **Approximation**

while $s_A \leq N_A$
$q = 0$, $i = 1$, $j = 2$, $p = m+2$
for $p > 4m/5$, $p = p-1$, $s_A = s_A + 1$
while $f_q < f_i + f_j$
for $j < m/2$, $j = j+1$, $s_A = s_A + 1$
for $i < j$, $i = i+1$, $s_A = s_A + 1$
end while
If $f_q - (f_i + f_j) = f_p$, Insert $f_j + f_p$ in P, delete f_q from Q
If $q < \frac{3m}{5}$
$q = q+1$
If f_q is in Q, $i = q+1$, $j = q+2$, $p = m+2$
end while
$P_0 = P_0 \cup P$, $P = Q \cup \{1, 2\}$

6. **Division**

while $s_D \leq N_D$
$m =$ (length of P) – 2 , $P = 1, 2, f_m, f_{m-1}, \ldots, f_2, f_1, f_0$
for $prime$ = 3,5,9 or 17
If f_0 is divisible by $prime$, insert $\frac{f_0}{prime}, \frac{2f_0}{prime}, \ldots, \frac{(prime-1)f_0}{prime}$ in P
Delete $\frac{f_0}{prime}, \frac{2f_0}{prime}, \ldots, \frac{(prime-1)f_0}{prime}, f_0$ from Q
$P_0 = P_0 \cup P$, $s_D = s_D + 1$
end while

7. **Halving**

while $s_H \leq N_H$
Arrange elements of Q in descending order as $q_0, q_1, \ldots, q_r$
$u = \lfloor \log(\frac{q_0}{q_r}) \rfloor$, $u_1 = \lfloor \frac{q_0}{2^u} \rfloor$

Insert $d = q_0 - u_1 2^u, q_0 - d = u_1 2^u, u_1 2^{u-1}, \ldots, 2u_1, u_1$ in P and
delete all them as well as q_0 from Q
If d and u_1 were not in P before the insertion, insert them in Q.
$s_H = s_H + 1$
$P_0 = P_0 \cup P$
$P = Q \cup \{1, 2\}$
end while

8. end while
9. Output P_0

4.2.3 Case III: Express larger elements in terms of the third largest element and every other element in terms of the smallest

Here the Makesequence employs two stages of one-thirding. In Stage I, the larger elements are expressed in terms of the third largest element. In Stage II, the remaining elements are expressed in terms of the smallest element.

Stage I:

1. Arrange all the elements of A in ascending order.
2. Add 1 and 2 to the sequence and get a Protosequence P in which we need to introduce more numbers to get the required addition chain.
3. Define $P = 1, 2, f_m, f_{m-1}, \ldots, f_2, f_1, f_0$, Q = elements in P which can not be expressed as sum of two smaller elements, $i = 0$, r = Number of elements in Q, $P' = \emptyset$
4. Update r and arrange elements of Q in descending order as $q_0, q_1, ..., q_r$

   while $i < 2$ and $Q \neq \emptyset$
   $u = \lfloor \log_3(\frac{q_i}{q_2}) \rfloor, u_1 = \lfloor \frac{q_i}{3^u} \rfloor$
   Insert $d = q_i - u_1 3^u, q_i - d = u_1 3^u, u_1 3^{u-1}, \ldots, 3u_1, u_1$ in P'
   $i = i + 1$
   end while

5. p = minimum element of P'
6. $A = A - P' \cup \{p\}$

Stage II:

1. Arrange all the elements of A in ascending order.

2. Add 1 and 2 to the sequence and get a Protosequence P in which we need to introduce more numbers to get the required addition chain.

3. Define $P = 1, 2, f_m, f_{m-1}, \ldots, f_2, f_1, f_0$, Q = elements in P which can not be expressed as sum of two smaller elements, $i = 0$, r = Number of elements in Q

4. while $i < r$ and $Q \neq \emptyset$
   Update r and arrange elements of Q in descending order as
   $q_0, q_1, \ldots, q_r$
   $u = \lfloor \log_3(\frac{q_i}{q_r}) \rfloor, u_1 = \lfloor \frac{q_i}{3^u} \rfloor$
   Insert $d = q_i - u_1 3^u, q_i - d = u_1 3^u, u_1 3^{u-1}, \ldots, 3u_1, u_1$
   in P and delete all of them as well as q_i from Q
   $i = i + 1$
   end while

5. Output $P \cup P'$

4.3 Ternary representation of the exponent

The ternary method employs a similar but different window algorithm. We consider two variants of the Makesequence, but the Window methods for both the algorithms are same. Among the two variants of Makesequence algorithms are employed here, the first Variant uses the usual Bos and Coster Makesequence heuristic. For the 2nd Variant, we proceed by representing larger elements in terms of smaller elements as the prefixes of the former.

The Window Method:

1. Express the number n in ternary form and define the ternary string as s.

2. Define a parameter k and a set A. User can specify the value of k on the basis of the value of n, $A = \emptyset$.

3. while the length of s is greater than k
   Define sub = sub-string of k characters of s from the left.
   Delete the zeroes, if any, in the right end of sub.
   $A = A \cup \{sub\}$
   Redefine s by deleting from it sub and the zeroes that follow it.
   end while

4. Delete the tail of zeroes from s.

5. $A = A \cup s$

An Illustrative example

$n = 26235947428953663183191,$
$s = 22212202221000222222210202101210202122002201 1022.$

$k = 4$

$\underline{2221}2202221000222222210202101210202122002201 1022$

$A = \{2221\}$
$s = 220222100022222221020210121020212200220110 22$

$A = \{\underbrace{22}_{8}, \underbrace{121}_{16}, \underbrace{221}_{25}, \underbrace{1022}_{35}, \underbrace{2021}_{61}, \underbrace{2201}_{73}, \underbrace{2202}_{74}, \underbrace{2221}_{79}, \underbrace{2222}_{80}\}$

4.3.1 Ternary I

Ternary I receives the window elements from the Window method described above. It employs three of the Bos and Coster Makesequence routines, namely Approximation, Division and Lucas. The Halfing routine is substituted by the one-thirding routine.

1. Arrange all the elements of A in ascending order. Add 1 and 2 to the sequence and get a Protosequence P in which we need to introduce more numbers to get the required addition chain.
2. Define $P = 1, 2, f_m, f_{m-1}, \ldots, f_2, f_1, f_0$, Q = elements in P which can not be expressed as sum of two smaller elements $P_0 = \emptyset$, $s_A = 0$, $s_D = 0$, $s_T = 0$, $s_L = 0$, N_A, N_D, N_T, N_L
3. while $Q \neq \emptyset$
4. **Lucas**

   while $s_L \leq N_L$
   Arrange elements of Q in descending order as $q_0, q_1, \ldots, q_r$
   for j=1 to r
   for i=0 to $j - 1$
   If $q_i - q_j$ is in P_0, delete q_i from Q
   If $j < r$, $j = j + 1$, $s_L = s_L + 1$
   If $i < r - 1$, $i = i + 1$, $s_L = s_L + 1$
   end while

5. **Approximation**

   while $s_A \leq N_A$
   $q = 0$, $i = 1$, $j = 2$, $p = m + 2$
   for $p > 4m/5$, $p = p - 1$, $s_A = s_A + 1$
   while $f_q < f_i + f_j$

for $j < m/2$, $j = j + 1$, $s_A = s_A + 1$
for $i < j$, $i = i + 1$, $s_A = s_A + 1$
end while
If $f_q - (f_i + f_j) = f_p$, Insert $f_j + f_p$ in P, delete f_q from Q
If $q < \frac{3m}{5}$
$q = q + 1$
If f_q is in Q, $i = q + 1$, $j = q + 2$, $p = m + 2$
end while

6. $P_0 = P_0 \cup P$, $P = Q \cup \{1, 2\}$
7. **Division**

while $s_D \leq N_D$
m = (length of P) – 2 , $P = 1, 2, f_m, f_{m-1}, \ldots, f_2, f_1, f_0$
for $prime$ = 3,5,9 or 17
If f_0 is divisible by $prime$, insert $\frac{f_0}{prime}, \frac{2f_0}{prime}, \ldots,$
$\frac{(prime-1)f_0}{prime}$ in P
Delete $\frac{f_0}{prime}, \frac{2f_0}{prime}, \ldots, \frac{(prime-1)f_0}{prime}, f_0$ from Q
$P_0 = P_0 \cup P$, $s_D = s_D + 1$
end while

8. **One thirding**

while $s_T \leq N_T$
Arrange elements of Q in descending order as $q_0, q_1, \ldots, q_r$
$u = \lfloor \log_3(\frac{q_0}{q_r}) \rfloor, u_1 = \lfloor \frac{q_0}{3^u} \rfloor$
Insert $d = q_0 - u_1 3^u, q_0 - d = u_1 3^u, u_1 3^{u-1}, \ldots, 3u_1, u_1$ in P
and delete all them as well as q_0 from Q.
If d and u_1 were not in P before the insertion, insert them in Q.
$s_T = s_T + 1$
$P_0 = P_0 \cup P$
$P = Q \cup \{1, 2\}$
end while

9. end while

4.3.2 Ternary II

In this variant of the Ternary Method, the set of window A sent by the Window Method are partitioned according to the window length, A_i being the partition containing windows of size i. Starting from windows of largest length, we search for the largest window which is a prefix of the former and store the leftover part of the former window in a suitable partition if that has size greater than the minimum window size i_{min}, else it is sent to a pool P. Once we have tested all the windows, B_i are defined by elements formed by suitable triplings of elements from various A_j where $i > j$ since we required these tripled elements to conjure up to the elements originally sent by the Window Method.

1. Partition the elements of A in sets A_i where i is the length of the string that expresses an element in A_i in ternary representation. $A = \cup_{i=i_{\min}}^{i_{\max}} A_i$ where $i_{\min} \leq i \leq i_{\max}$.
2. $P = \emptyset$
3. for $i = i_{\max}$ to $i_{\min} + 1$ by -1
4. for $j = i - 1$ to $i_{\min}$ by -1
5. For all elements $x \in A_i$ with prefix $y \in A_j$ truncate y from x to get z and send z to A_{i-j} if $i-j > i_{\min}$, else send z to P. Any substring of zeroes in the left end of z is deleted before it is sent to P.
6. If $i_{\min} > 1, P$ can be non-null. If x_1 is the largest element in P, the output is $\{1, 2, \ldots, x_1\} \cup (\cup_{i=i_{\min}}^{i_{\max}} A_i) \cup (\cup_{i=i_{\min}}^{i_{\max}} B_i)$ where the B_i are formed by the triplings and additions of various elements in A_i to add up to the elements in A.

An illustrative example

The Window Method sent $A = \{\underbrace{22}_{8}, \underbrace{121}_{16}, \underbrace{221}_{25}, \underbrace{1022}_{35}, \underbrace{2021}_{61}, \underbrace{2201}_{73}, \underbrace{2202}_{74}, \underbrace{2221}_{79}, \underbrace{2222}_{80}\}$

Here, $i_{\max} = 4, i_{\min} = 2$, and the initial partitions are $A_2 = \{22\}, A_3 = \{121, 221\}$, $A_4 = \{2221, 2202, 2222, 2021, 2201, 1022\}$

At the end of the iterations we get $P = \{1, 2\}$ and hence $x_1 = 2$
We also get $A_2 = \{22,21\}$, $A_3 = \{121,221\}$, $A_4 = \{2221,2202,2222,2021,2201,1022\}$ $B_2 = \{10,20\}$, $B_3 = \{100, 200, 220\}$, $B_4 = \{1000,2000,2200\}$

Hence the Output Makesequence is $\underbrace{1}_{1}, \underbrace{2}_{2}, \underbrace{10}_{3}, \underbrace{20}_{6}, \underbrace{21}_{7}, \underbrace{22}_{8}, \underbrace{100}_{9}, \underbrace{200}_{18}$, $\underbrace{220}_{24}, \underbrace{121}_{16}, \underbrace{221}_{25}, \underbrace{1000}_{27}, \underbrace{1022}_{35}, \underbrace{2000}_{54}, \underbrace{2200}_{72}, \underbrace{2201}_{73}, \underbrace{2202}_{74}, \underbrace{2221}_{79}, \underbrace{2021}_{61}, \underbrace{2222}_{80}$

Chapter 5

Conclusion

The dissertation work had two components. Initially we analyzed the Brauer's algorithm, modified Brauer's algorithm and Yao's algorithm whose results are available in 5.1. At a later stage we incorporated three modifications of the Bos and Coster algorithm and compared their efficiency with the original model taking a large exponent used by Bos and Coster [3] as a test case. We present this comparative study in 5.2.

5.1 Comparison between Brauer's, modified Brauer's and Yao's algorithms

The Brauer's algorithm, modified Brauer's algorithm and Yao's algorithm has been coded in C language. The results obtained for various n and k are shown in the Table 1 of the Appendix. From $n = 1000$ to $n = 2100000000$ we studied the length of the Brauer's chain, Modified Brauer's chain and Yao's chain for k =2, 3, 4 and 5.

Some interesting results:

1. For the Modified Brauer's Chain, the length is minimum for $k = 2$ when $n \leq$10000000.For a value of n somewhere between 10000000 and 10500000, the optimum value of k changes from 2 to 3.
2. For smaller values of n, the ratio of the length of Brauer's chain to the length of Modified Brauer's chain is high. As n increases, this ratio decreases considerably.
3. For smaller n and larger k, the Yao's chain is smaller than the Brauer's chain. For larger n, the Brauer's chain is smaller than the Yao's chain.
4. The Modified Brauer's Chain is consistently smaller than the Brauer's as well as the Yao's chain though the gap is small with the Yao's chain for smaller n and larger k.

5.2 Comparison of the efficiency of our modifications of Bos and Coster algorithm with the original algorithm

The test case we worked with was the exponent 10110001110010000001110100101

00111010100000010111100000111110011001010101111.
The length of the addition chain for this number using Bos and Coster Algorithm is 89, of which length of sequence that computes intermediates was 22, number of squaring needed was 62 and the number of multiplies for intermediates was 5 . Following are the analysis and the results for the different algorithms we devised.

5.2.1 Pure DBNS

The window elements in DKS double base representation are

5689 = 5832 - 144 + 1
933 = 972 - 36 - 3
117 = 144 – 27
47 = 48 – 1
499 = 432 + 64 + 3
343 = 324 + 16 + 3

This requires a make-sequence 1, 2, 3, 4, 8, 9, 16, 18, 27, 36, 48, 47, 32, 64, 67, 19, 144, 81, 162, 243, 324, 432, 499, 343, 972, 2916, 5832, 5833, 5689 of length 28, which is worse by 6 from the original algorithm by Bos and Coster

5.2.2 One thirding

Express every other element in terms of the smallest

Example: 1, 2, 4, 8, 16, 24, 48, 47, 49, 94, 86, 87, 188, 141, 117, 282, 351, 343, 484, 500, 499, 564, 613, 846, 933, 1692, 5076, 5689

The length of this chain is 27, which is worse by 5 from the original algorithm by Bos and Coster

Express larger elements in terms of the third largest element

Example: 1, 2, 3, 4, 8, 16, 32, 48, 47, 31, 78, 70, 117, 141, 234, 265, 343, 311, 933, 1198, 530, 499, 1497, 4491, 5689

The length of this chain is 24.

Combination of the earlier two strategies

Example: 1, 2, 3, 4, 6, 7, 8, 24, 48, 47, 94, 91, 87, 16, 75, 76, 70, 117, 141, 423, 499, 846, 933, 2799, 5598, 5689

The length of this chain is 25.

5.2.3 Ternary representation

For the given number, we give the decimal, binary and ternary representations.

Decimal: 26235947428953663183191

Binary: 10110001110010000001110100101001110101

00000010111100000111110011001010101111

Ternary: 22212202221000222222210202101210202122002201 1022

A window representation of the ternary number is
22212202221000222222210202101210202122002201 1022

11=4, 22=8, 11022=116, 2021=61, 202122=557, 121=16, 2221=79,

2222=80, 222122=719

The make-sequence chain is 1, 3, 4, 7, 8, 9, 18, 27, 54, 12, 36, 61, 79, 108, 116, 183, 237, 549, 557, 711, 719 which has length 20.

For this representation, we need 42 triplings, 20 additions and 9 intermediate additions , i.e., 71 operations which is an improvement from Bos and Coster's methods which required 89 operations.

5.3 Scope for future work

The superiority of the heuristics suggested in this work requires conclusive establishment by extensive experiments performed on randomly chosen values of n. It can also be an interesting study to find out if the use of DBNS along with the four original routines of the Makesequence algorithm proposed by Bos and Coster [BosCoster90] should provide improvement over the Bos and Coster Algorithm.

Bibliography

[Erdos60] P. Erdos, Remarks on number theory. III. On addition chains, Acta Arith. (1960), 77–81.

[Knuth81] D. E. Knuth, "Seminumerical Algorithms," 2nd ed., "The Art of Computer Programming,"Vol. 2, Addison-Wesley, Reading, MA, 1981.

[BosCoster90] J. Bos and M. Coster, Addition chain heuristics, Advances in Cryptology—Proceedings of Crypto '89, Vol. 435, pp. 400–407, Springer-Verlag, Berlin/New York, 1990.

[Dimitrov95] V. S. Dimitrov and T. Cooklev. Hybrid Algorithm for the Computation of the Matrix Polynomial $I + A + ... + A^{N-1}$. IEEE Trans. on Circuits and Systems, 42(7):377–380, 1995.

[Imbert2005] V. S. Dimitrov, L. Imbert, and P. K. Mishra. Efficient and Secure Elliptic Curve Point Multiplication Using Double-Base Chains, Advances in Cryptology – Asiacrypt 2005, volume 3788 of Lecture Notes in Comput. Sci., pages 59–78. Springer.

[Bernstein2011] Daniel J. Bernstein, Pippenger's exponentiation algorithm, URL: http://cr.yp.to/papers/pippenger.pdf

[LempelZiv76] A. Lempel and J. Ziv, On the complexity of finite sequences, IEEE Trans. Inform. Theory, IT-22, (1976)

[Yacovi99] Yacov Yacobi, Fast Exponentiation Using Data Compression, SIAM Journal on Computing, Volume 28 , Issue 2 (April 1999), Pages: 700 - 703

[Doche2009] Christophe Doche, David R. Kohel and Francesco Sica, Double-Base Number System for Multi-Scalar Multiplications, Proceeding EUROCRYPT '09 Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques

[Gordon98] Daniel M.Gordon, "A Survey of Fast Exponentiation Methods", Journal of Algorithms 27, pp.129-146, 1998.

[Imai2011] Vorapong Suppakitpaisarn, Masato Edahiro and Hiroshi Imai, "Fast Elliptic Curve Cryptography Using Optimal Double-Base Chains", URL: http://eprint.iacr.org/2011/030

Appendix A

Appendix

Table A.1: Comparison of the length of the Brauer's chain, Modified Brauer's chain and Yao's chain for various n and k

n	k	Brauer's	Modified Brauer's	Yao's
1000	3	21	13	21
1000	4	24	15	24
1000	5	36	22	22
1000	2	14	13	16
2000	2	17	14	17
2000	3	18	14	21
2000	4	24	16	22
2000	5	42	23	28
10000	2	20	17	20
10000	3	22	17	23
10000	4	29	20	23
10000	5	42	26	30
100000	2	26	20	24
100000	3	26	20	26
100000	4	34	23	33
100000	5	48	30	30
1000000	2	29	24	27
1000000	3	30	24	32
1000000	4	34	26	32
1000000	5	48	33	38
10000000	2	35	29	32
10000000	3	34	29	37
10000000	4	39	31	38
10000000	5	54	37	45
100000000	2	41	34	39
100000000	3	38	33	46
100000000	4	44	35	46
100000000	5	60	41	56
1000000000	2	44	38	42
1000000000	3	42	36	50
1000000000	4	49	38	56
1000000000	5	60	45	62
2100000000	2	47	41	NAN
2100000000	3	46	38	NAN
2100000000	4	49	40	NAN
2100000000	5	66	46	NAN