Advances in Alternative Non-adjacent Form Representations

Gildas Avoine[⋆], Jean Monnerat[⋆⋆], and Thomas Peyrin

EPFL
Lausanne, Switzerland

Abstract. From several decades, *non-adjacent form* (NAF) representations for integers have been extensively studied as an alternative to the usual binary number system where digits are in $\{0,1\}$. In cryptography, the *non-adjacent digit set* (NADS) $\{-1,0,1\}$ is used for optimization of arithmetic operations in elliptic curves. At SAC 2003, Muir and Stinson published new results on alternative digit sets: they proposed infinite families of integers x such that $\{0,1,x\}$ is a NADS as well as infinite families of integers x such that $\{0,1,x\}$ is not a NADS, so called a NON-NADS. Muir and Stinson also provided an algorithm that determines whether x leads to a NADS by checking if every integer $n \in [0, \lfloor \frac{-x}{3} \rfloor]$ has a $\{0,1,x\}$-NAF. In this paper, we extend these results by providing *generators* of NON-NADS infinite families. Furthermore, we reduce the search bound from $\lfloor \frac{-x}{3} \rfloor$ to $\lfloor \frac{-x}{12} \rfloor$. We introduce the notion of *worst* NON-NADS and give the complete characterization of such sets. Beyond the theoretical results, our contribution also aims at exploring some algorithmic aspects. We supply a much more efficient algorithm than those proposed by Muir and Stinson, which takes only 343 seconds to compute all x's from 0 to -10^7 such that $\{0,1,x\}$ is a NADS.

1 Introduction

It is well known that every positive integer n can be represented as a finite sum of the form $\sum_{i=0}^{n} a_i 2^i$, denoted by $(\ldots a_3 a_2 a_1 a_0)_2$, where the *digits* a_i's are picked in the *digit set* $D = \{0,1\}$. Using the digit set $\{0,1\}$ is a common way to represent integers but for some efficiency purposes some alternative digit sets have been proposed during the last decades.

Ternary representations (with radix 3) are mainly due to Lalanne [10] but took off in 1951 when Booth [1] proposed a fast technique to compute the representation of the product of two integers using the $\{-1,0,1\}$ radix 2 representation. In 1960, Reitwiesner [17] proved that every integer has a canonical $\{-1,0,1\}$

[⋆] Supported in part by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

[⋆⋆] Supported in part by a grant of the Swiss National Science Foundation, 200021-101453/1.

radix 2 representation with a minimal number of nonzero digits. This representation called *non-adjacent form* (NAF) is obtained if for any two adjacent digits at least one is zero. Later, in 1989, Jedwab and Mitchell [6] presented an interesting cryptographic application of such representations showing that using the digit set $\{-1, 0, 1\}$ can reduce the number of multiplications in the square-and-multiply algorithm for exponentiation. In elliptic curves, where inversion can be done for (almost) free, exponentiations are much more efficient with such representations. Using this property, Morain and Olivos [14] proposed in 1991 an algorithm speeding up operations over elliptic curves using the $\{-1, 0, 1\}$ digit set. More recently Joye and Tymen [7] proposed a compact encoding of non-adjacent forms applied to elliptic curves, in particular to the Koblitz curves. During the last decade, a certain amount of work has been devoted to non-adjacent form representations such as [2, 3, 4, 5, 8, 9, 12, 13, 18, 19].

In our case, we focus on *ternary* radix 2 representations using the digit set $\{0, 1, x\}$ where x is a negative integer. Determining which sets $\{0, 1, x\}$ provide non-adjacent forms for every positive integer is still an open problem. Such sets are called *non-adjacent digit sets* (NADS). Muir and Stinson [15, 16] gave new results at SAC 2003, proposing some properties that x must verify in order to lead to a NADS and they gave some infinite families of x such that $\{0, 1, x\}$ is or is not a NADS. In the latter case, we say that $\{0, 1, x\}$ is a NON-NADS. They also provided an algorithm that determines whether x is a NADS by checking whether every integer $n \in [0, \lfloor \frac{-x}{3} \rfloor]$ has a $\{0, 1, x\}$-NAF.

We extend in this paper their results by proposing generators that produce infinite families of NON-NADS as much as we wish and we give a way to determine such generators. We reduce also the search bound from $\lfloor \frac{-x}{3} \rfloor$ to $\lfloor \frac{-x}{12} \rfloor$. We introduce the notion of *worst* NON-NADS and give a complete characterization of these numbers. Our contribution aims also at exploring algorithmic aspects related to NADS. So, we propose some improvements of the Muir and Stinson's algorithms [15, 16] that comes from our new theoretical results and we propose a new approach to compute NADS. The first algorithm proposed in [15] took about one day in order to find all x's from -1 to -10^7 such that $\{0, 1, x\}$ is a NADS. While an improved version also due to Muir and Stinson [16] takes about 20 minutes, our own algorithm takes only 343 seconds.

2 Preliminaries and Previous Works

2.1 Definitions and Notation

Every positive integer n can be represented as a finite sum of the form $\sum_{i=0}^{n} a_i 2^i$, denoted by $(\ldots a_3 a_2 a_1 a_0)_2$. Here, the *digits* a_i's are in the *digit set* $D = \{0, 1\}$.

Definition 1. *The Hamming weight of a non-negative integer n, denoted by $w(n)$, is the number of ones in the usual $\{0, 1\}$-radix 2 representation of n.*

Definition 2. *The length of a radix 2 representation* $(\dots a_2 a_1 a_0)_2$ *is the largest integer* ℓ *such that* $a_{\ell-1} \neq 0$ *but* $a_i = 0$ *for all* $i \geq \ell$.

The set of all strings of digits from D is denoted by D^* and contains the empty string ϵ. Every D-radix 2 representation matches a string in D^* and every string in D^* matches a D-radix 2 representation. For $\alpha, \beta \in D^*$, we denote their concatenation by $\alpha \| \beta$. The terminology for representations can be applied to strings. We note $\widehat{\alpha}$ the string formed by deleting the leading zeros from α. For a given digit set D and an integer n, we define the map $R_D(n)$ such that $R_D(n) = \widehat{\alpha}$ where $\alpha \in D^*$ is a D-NAF for n, if one exists, and $R_D(n) = \perp$ otherwise. Here $\perp$ represents the symbol not in D. We are interested in determining which integers have D-NAF's, so we define the set $\text{NAF}(D) := \{n \in \mathbb{Z} : R_D(n) \neq \perp\}$.

Definition 3. *D is a nonadjacent digit set if* $\mathbb{Z}^+ \subseteq$ *NAF(D).*

2.2 Characterization of NADS

First of all, we give a few theorems, whose proofs can be found in [15], giving necessary conditions for $\{0, 1, x\}$ to be a NADS or a NON-NADS.

Theorem 1. *Let* $D = \{0, 1, x\}$. *If there exists* $n \in$ *NAF(D) with* $n \equiv 3 \pmod 4$, *then* $x \equiv 3 \pmod 4$.

Theorem 2. *The only NADS of the form* $\{0, 1, x\}$ *with* $x > 0$ *is* $\{0, 1, 3\}$.

Theorem 3. *If* $x \equiv 3 \pmod 4$, *then any integer has at most one finite length* $\{0, 1, x\}$*-NAF form with no leading zeros.*

From Theorems 1 to 3 we see that a $\{0, 1, x\}$-NAF is unique and that x must be negative and congruent to 3 modulo 4 for $\{0, 1, x\}$ to be a NADS (except for $D = \{0, 1, 3\}$). Hence, we only consider NADS such that $x < 0$. The following lemmas lead to an algorithm that determines whether an integer $n \in \mathbb{Z}^+$ has a $\{0, 1, x\}$-NAF.

Lemma 1. *If* $n \equiv 0 \pmod 4$ *then* $n \in$ *NAF(D) if and only if* $n/4 \in$ *NAF(D). Further, if* $n \in$ *NAF(D) then* $R_D(n) = R_D(\frac{n}{4}) \| 00$.

Lemma 2. *If* $n \equiv 1 \pmod 4$ *then* $n \in$ *NAF(D) if and only if* $(n-1)/4 \in$ *NAF(D). Further, if* $n \in$ *NAF(D) then* $R_D(n) = R_D(\frac{n-1}{4}) \| 01$.

Lemma 3. *If* $n \equiv 2 \pmod 4$ *then* $n \in$ *NAF(D) if and only if* $n/2 \in$ *NAF(D). Further, if* $n \in$ *NAF(D) then* $R_D(n) = R_D(\frac{n}{2}) \| 0$.

Lemma 4. *If* $n \equiv 3 \pmod 4$ *then* $n \in$ *NAF(D) if and only if* $(n-x)/4 \in$ *NAF(D). Further, if* $n \in$ *NAF(D) then* $R_D(n) = R_D(\frac{n-x}{4}) \| 0x$.

We define now the function $f_D : \mathbb{N} \to \mathbb{N}$ as follows: $f_D(n) = \frac{n}{4}$ if $n \equiv 0 \pmod 4$, $f_D(n) = \frac{n-1}{4}$ if $n \equiv 1 \pmod 4$, $f_D(n) = \frac{n}{2}$ if $n \equiv 2 \pmod 4$, $f_D(n) = \frac{n-x}{4}$ if $n \equiv 3 \pmod 4$. For the sake of simplicity, we abuse the notation by denoting $f_0(n) = \frac{n}{4}$, $f_1(n) = \frac{n-1}{4}$, $f_2(n) = \frac{n}{2}$, and $f_3(n) = \frac{n-x}{4}$. Note that $\forall n \in [0, \frac{x}{3}]$, $f_0(n) < n$, $f_1(n) < n$, $f_2(n) < n$, and $f_3(n) > n$. We denote f^i the i-fold composition of the function f. We introduce now the *graph* G_n of an integer n for a given digit set $\{0, 1, x\}$, whose vertices are the iterations of the function f_D on n:

$$n \longrightarrow f_D(n) \longrightarrow f_D^2(n) \longrightarrow f_D^3(n) \longrightarrow \dots$$

where either $\exists k \geq 0$ such that $f_D^k(n) = 0$ or $\exists k_1, k_2,\ 0 \leq k_1 \leq k_2$ such that $f_D^{k_1}(n) = f_D^{k_2}(n)$. In other words, either G_n is a path terminating at 0, or G_n contains a directed cycle of integers in the interval $\{0, 1, 2, \dots, \lfloor\frac{-x}{3}\rfloor\}$ as proved hereafter. The *length of the cycle* is defined as $k_2 - k_1$. Every vertex of G_n is positive. Suppose that $f_D^i(n) < \frac{-x}{3}$, we prove that $f_D^{i+1}(n) < \frac{-x}{3}$. If $f_D^i(n) \equiv 0, 1, 2 \pmod 4$ we have $f_D^{i+1}(n) \leq f_D^i(n) < \frac{-x}{3}$. If $f_D^i(n) \equiv 3 \pmod 4$ then

$$\begin{aligned} f_D^i(n) < \frac{-x}{3} &\implies \frac{f_D^i(n) - x}{4} < \frac{\frac{-x}{3} - x}{4} \\ &\implies f_D^{i+1}(n) < \frac{-x - 3x}{12} = \frac{-x}{3}. \end{aligned}$$

By extension, we define the *graph* $G(x)$ of an integer x as $G(x) := \bigcup_{n=0}^{\lfloor\frac{-x}{3}\rfloor} G_n$. Note that if $\{0, 1, x\}$ is a NADS, then $G(x)$ is a directed tree whose root is 0. We define now the function $g_D : \mathbb{N} \to D^*$ such that: $g_D(n) =$ "00" if $n \equiv 0 \pmod 4$, $g_D(n) =$ "01" if $n \equiv 1 \pmod 4$, $g_D(n) =$ "0" if $n \equiv 2 \pmod 4$, $g_D(n) =$ "0x" if $n \equiv 3 \pmod 4$. From Lemmas 1 to 4 and the definitions of f_D and g_D, Muir and Stinson proposed Lemma 5 that yields Algorithm 1.

Algorithm 1: NAF(n,x)

```
$\alpha \leftarrow \epsilon$
while $n > \frac{-x}{3}$
    do { $\alpha \leftarrow g_D(n) \| \alpha$
         $n \leftarrow f_D(n)$

$S \leftarrow \varnothing$
while $n \neq 0$
    do { if $n \in S$
            then return $\perp$
         $S \leftarrow S \cup \{n\}$
         $\alpha \leftarrow g_D(n) \| \alpha$
         $n \leftarrow f_D(n)$

return $\hat{\alpha}$
```

Algorithm 2: Is-NADS?(x)

```
$N \leftarrow 3$

while $N \leq \frac{-x}{3}$
    do { $n \leftarrow N$
         $S \leftarrow \varnothing$
         while $n \neq 0$
             do { if $n \in S$
                     then return false
                  $S \leftarrow S \cup \{n\}$
                  $n \leftarrow f_D\ (n)$
         $N \leftarrow N + 4$

return true
```

Lemma 5. *For any $n \in \mathbb{N}$, $n \in$ NAF(D) if and only if $f_D(n) \in$ NAF(D). Further, if $n \in$ NAF(D) then $R_D(n) = R_D(f_D(n)) \| g_D(n)$.*

The time complexity of the NAF algorithm in the worst case is straightforward: the complexity of the first loop is $O(\log n)$ while the complexity of the second one is $O(|x|)$. The complexity of the algorithm, in the worst case, is therefore $O(\log n + |x|)$. We expose now Theorem 4 that provides algorithm Is-NADS? (See Algorithm 2) determining whether or not a given $x < 0$ leads to a NADS.

Theorem 4. *Suppose x is a negative integer and $x \equiv 3 \pmod 4$. If every element in the set $\{n \in \mathbb{Z}^+ : n \leq \lfloor \frac{-x}{3} \rfloor, n \equiv 3 \pmod 4\}$ has a $\{0,1,x\}$-NAF, then $\{0,1,x\}$ is a NADS.*

The algorithm Is-NADS? requires $O(|x|)$ tests (one test is roughly equivalent to the second loop of the algorithm NAF), therefore the complexity of Is-NADS? is $O(|x|^2)$. Finally, Muir and Stinson [15] give some characterizations of infinite families of NADS and NON-NADS. Among them, we will use the two following theorems.

Theorem 5. *Let x be a negative integer with $x \equiv 3 \pmod 4$. If $(2^s - 1) \mid x$ for any $s \geq 2$, then $\{0,1,x\}$ is not a NADS.*

Theorem 6. *Let x be a negative integer with $x \equiv 3 \pmod 4$. If $(4 \cdot m_i - 1) < -x < (3 \cdot 2^i)$ for some $i \geq 0$, then $\{0,1,x\}$ is not a NADS, where*

$$m_i := \begin{cases} 2 \cdot \frac{2^i - 1}{3} & \text{for } i \text{ even} \\ \frac{2^{i+1}-1}{3} & \text{for } i \text{ odd.} \end{cases}$$

3 New Theoretical Results

3.1 Improvement on the Search Domain

By Theorem 4, we know that determining whether $\{0,1,x\}$ is a NADS can be performed by checking whether every element of the set $\{n \in \mathbb{Z}^+ : n \leq \lfloor \frac{-x}{3} \rfloor, n \equiv 3 \pmod 4\}$ has a $\{0,1,x\}$-NAF. Here, we prove that the search bound $\lfloor \frac{-x}{3} \rfloor$ can be improved when 3 or/and 7 do not divide x. So, Theorem 7 reduces the bound to $\lfloor \frac{-x}{6} \rfloor$ and Theorem 8 goes further reducing the search domain to $]0, \lfloor \frac{-x}{12} \rfloor] \cup [\lfloor \frac{-x}{7} \rfloor, \lfloor \frac{-x}{6} \rfloor]$.

Theorem 7. *Let x be a negative integer such that $3 \nmid x$ and $x \equiv 3 \pmod 4$. If every element in the set $\{n \in \mathbb{Z}^+ : n \leq \lfloor \frac{-x}{6} \rfloor, n \equiv 3 \pmod 4\}$ has a $\{0,1,x\}$-NAF, then $\{0,1,x\}$ is a NADS.*

Proof. Let $n \equiv 3 \pmod 4$ be a positive integer such that $n \leq \lfloor \frac{-x}{3} \rfloor$. Since $3 \nmid x$, $n < \frac{-x}{3}$. We have to show that G_n must contain at least one vertex which is less than $\frac{-x}{6}$. In other words, this corresponds to show the existence of $j \in \mathbb{N}$ satisfying $f_D^j(n) < \frac{-x}{6}$. From definition of f_D, we also remark that if $f_D^i(n) < \frac{-x}{3}$ is congruent to $0, 1, 2$ modulo 4, then $f_D^{i+1}(n) < \frac{-x}{6}$. So, it remains to show that $f_D^i(n)$ cannot be congruent to 3 modulo 4 for all $i \in \mathbb{N}$. From Section 2.2, we know that $f_D^i(n) < \frac{-x}{3} \Rightarrow f_D^{i+1}(n) < \frac{-x}{3}$ for $i \in \mathbb{N}$ and that f_D is strictly increasing on the upper bounded set $\{n \in \mathbb{Z}^+ : n \leq \frac{-x}{3}, n \equiv 3 \pmod 4\}$. Hence, $f_D^i(n)$ cannot be congruent to 3 modulo 4 for all $i \in \mathbb{N}$. □

We give here a further improvement on the search domain.

Theorem 8. *Let x be a negative integer such that $3 \nmid x$, $7 \nmid x$ and $x \equiv 3 \pmod 4$. If every element in the set $\{n \in \mathbb{Z}^+ : n \leq \lfloor \frac{-x}{12} \rfloor, n \equiv 3 \pmod 4\} \bigcup \{n \in \mathbb{Z}^+ : \lfloor \frac{-x}{7} \rfloor \leq n \leq \lfloor \frac{-x}{6} \rfloor, n \equiv 3 \pmod 4\}$ has a $\{0, 1, x\}$-NAF, then $\{0, 1, x\}$ is a NADS.*

Proof. Let n be a positive integer such that $n \equiv 3 \pmod 4$ and $\lfloor \frac{-x}{12} \rfloor \leq n \leq \lfloor \frac{-x}{7} \rfloor$. We will show that G_n contains at least a vertex that lies in the interval $[\frac{-x}{7}, \frac{-x}{6}]$ or that is less than $\frac{-x}{12}$. First, we notice that if an element of G_n is congruent to 0 or 1 modulo 4, then this one will be sent to an integer less than $\frac{-x}{12}$ since this element cannot be greater than or equal $\frac{-x}{3}$. So, it remains to consider the n's for which G_n contains only vertices congruent to 2 or 3 modulo 4. Given that $f_2 \circ f_2 = f_0$, such a n is transformed by iterations of the form

$$f_2 \circ f_3^{i_k} \circ f_2 \circ f_3^{i_{k-1}} \circ \cdots f_2 \circ f_3^{i_1}, \tag{1}$$

for some integer $k \geq 1$ and where $i_1, \ldots, i_k$ are positive integers. We set $F := f_2 \circ f_3$. We see that $F(n) = \frac{n-x}{8}$ and that $F(n) > n \Leftrightarrow n < \frac{-x}{7}$. From the properties of the function f_3, we can conclude that $f_2 \circ f_3^\ell(n) \geq F(n)$ for $\ell \in \mathbb{N}$ and $n \leq \frac{-x}{3}$. Hence, the value of some iterations of the form (1) applied to n is greater or equal to $F^k(n)$. We finally deduce that there exists a positive integer k such that the resulting integer n' of the iteration (1) applied on n is greater than $\frac{-x}{7}$, since the intermediate value increases after each iteration of a function of the form $f_2 \circ f_3^\ell$ and since $7 \nmid x$. Moreover, n' is less than $\frac{-x}{6}$ since every values of G_n are less than $\frac{-x}{3}$ and that the last operation of (1) is a division by 2.

□

Conjecture 1. Let x be a negative integer such that $3 \nmid x$, $7 \nmid x$ and $x \equiv 3 \pmod 4$. If every element in the set $\{n \in \mathbb{Z}^+ : n \leq \lfloor \frac{-x}{12} \rfloor, n \equiv 3 \pmod 4\}$ has a $\{0, 1, x\}$-NAF, then $\{0, 1, x\}$ is a NADS.

The new results presented in this section are particularly important because they allow to reduce significantly the running time of Algorithm 2, as we will see in Section 4.

3.2 Generators of Infinite Families of NON-NADS

In this section, we present a way to generate as many NON-NADS families as we want. From a theoretical point of view, this method allows to find all NON-NADS. In practice, it will be used as a trade-off in our algorithm Find-NADS (See Section 4) that computes every x such that $\{0,1,x\}$ is a NADS.

The idea of our method comes from the fact that $n \in \mathrm{NAF}(D)$ if and only if G_n does not contain any directed cycle. So, the existence of an integer n such that G_n contains a directed cycle implies that D is not a NADS. Instead of looking for a criteria on x for which there exists such a n, we consider a cycle of a given form and deduce the values x for which n lies in this cycle. More precisely, we choose the length t of the cycle as well as the sequence of the t different functions f_i for $i \in \{0,1,2,3\}$ that are applied successively on n. Once the form of the cycle is chosen, we set for a positive integer $n \equiv 3 \pmod 4$ the equation

$$f_D^t(n) = f_{i_t} \circ f_{i_{t-1}} \circ \ldots f_{i_1}(n) = n, \tag{2}$$

where $i_k \in \{0,1,2,3\}$ for $k = 1,2\ldots,t$. We denote such a cycle of length t as $i_1|i_2|\ldots|i_t$. From (2), we obtain a relation of the form $c_1 n = c_2 x$ for two given $c_1, c_2 \in \mathbb{Z}$. It remains to substitute $n = 4k-1$ in this equation and solve it with the conditions that $k \in \mathbb{N}$ and x is negative with $x \equiv 3 \pmod 4$. Note that $i_1 = 3$.

2-Cycles. To illustrate our method, we show how we can concretely find every cycle of length 2. Such a cycle is called a 2-cycle. First, we observe that we have 3 possible 2-cycles, namely 3|0, 3|1 and 3|2. They correspond to the equations $\frac{n-x}{16} = n$, $\frac{n-x-4}{16} = n$ and $\frac{n-x}{8} = n$. The first equation provides $x = -15n$ and since $n = 4k-1$, we then have $x = -60k+15$ for $k \in \mathbb{N}$. By setting $k = 7$, we see that $n = 27$, $x = -405$ and $f_3(27) = 108 = 4 \cdot 27$. The second equation provides $x = -15n - 4 = -60k + 11$. Similarly, we obtain $x = -28k + 7$ from the third equation.

Theorem 9. *If $x = -60k + 15$, $x = -60k + 11$ or $x = -28k + 7$ with $k \in \mathbb{N}$, then $\{0,1,x\}$ is a NON-NADS.*

Some Cycles of Arbitrary Length. Here, we apply our method to find an infinite number of NON-NADS families. As an illustration, we look for the x's whose graph $G(x)$ contain a cycle of the form $3|3|3|\ldots|3|0$. Let $t \geq 2$ be the length of this cycle, we have to solve $f_0 \circ f_3^{t-1}(n) = n$. Let us first compute $f_3^{t-1}(n)$. We have

$$f_3^{t-1}(n) = \frac{n - x\sum_{i=1}^{t-1} 4^{i-1}}{4^{t-1}} = \frac{n - \frac{x(4^{t-1}-1)}{3}}{4^{t-1}}$$

and hence we get the equation

$$f_0 \circ f_3^{t-1}(n) = \frac{n - \frac{x(2^{2t-2}-1)}{3}}{2^{2t-1}} = n.$$

This holds if and only if $-x(2^{2t-2}-1)/3 = n(2^{2t-1}-1)$. From this, x has to be a multiple of $(2^{2t-1}-1)$ since $\gcd((2^{2t-2}-1)/3, 2^{2t-1}-1) = 1$. Moreover, $x \equiv 3 \pmod 4$ implies that x is of the form $x = -(4k-1)(2^{2t-1}-1)$ for $k \in \mathbb{N}$. We can also see that $n = (4k-1)(4^{t-1}-1)/3$ is congruent to 3 modulo 4 and that it is positive.

Theorem 10. *Let $t \geq 2$ and $k > 0$ be two integers and $x = -(4k-1)(2^{2t-1}-1)$. Then $\{0,1,x\}$ is a NON-NADS.*

Note that for $t = 2$, this generates the x-family corresponding to that of Theorem 9, namely $-28k+7$. Obviously, if we consider cycles of another form (instead of $3|3|3|\ldots|3|0$) we obtain some other generators.

3.3 Worst NON-NADS

We introduce in this section the notion of *worst NON-NADS* and give a complete characterization of it.

Definition 4. *Let x be a negative integer such that $x \equiv 3 \pmod 4$. $\{0,1,x\}$ is a worst NON-NADS if for all $n \leq -\frac{x}{3}$ with $n \equiv 3 \pmod 4$, $n \notin NAF(\{0,1,x\})$.*

Theorem 11. *Let x be a negative integer such that $x \equiv 3 \pmod 4$. $\{0,1,x\}$ is a worst NON-NADS if and only if there exists $i \geq 2$ such that $(4m_i - 1) < -x < (3 \cdot 2^i)$, where*

$$m_i := \begin{cases} 2 \cdot \frac{2^i-1}{3} & \text{for } i \text{ even} \\ \frac{2^{i+1}-1}{3} & \text{for } i \text{ odd} \end{cases}$$

Proof. We first prove that if a given x is in an interval of the form $]-3\cdot 2^i, 1-4m_i[$ then $\{0,1,x\}$ is a worst NON-NADS. Next we prove that if $\{0,1,x\}$ is a worst NON-NADS then it is in such an interval. Such an interval is called a *gap*. The first part of the proof directly comes from the proof of Theorem 21 of [16].

We prove now the converse statement. In other words, we show that for each x which is not in a gap, there exists a n such that $n \in \mathrm{NAF}(\{0,1,x\})$. We introduce the notion of *pivot*: x_p is a pivot if there exists a $i \geq 2$ s.t. $x_p = 3 - 2^{i+2}$.

We prove that, for all $i \geq 2$, there is no worst NON-NADS in $[3-2^{i+2}, -3\cdot 2^i - 1]$. Let x_p be the pivot $3 - 2^{i+2}$; we have

$$f_{\{0,1,x_p\}}(3) = \frac{3-(3-2^{i+2})}{4} = 2^i,$$

implying that $3 \in \mathrm{NAF}(\{0,1,x_p\})$. We have furthermore $f_{\{0,1,x_p+4k\}}(3+4k) = f_{\{0,1,x_p\}}(3)$ for all integers $k \geq 1$. Therefore, $x_p + 4k$ is a worst NON-NADS if

$$3 + 4k < \left\lfloor \frac{-x_p - 4k}{3} \right\rfloor < \frac{-x_p - 4k}{3} - 1.$$

Hence, we can compute that this inequality holds if and only if $k < 2^{i-2} - \frac{15}{16}$. This implies that for every $x \in [3 - 2^{i+2}, -3 \cdot 2^i - 1]$, we have found an $n < \lfloor -\frac{x}{3} \rfloor$ that is a $\{0, 1, x\}$-NAF.

It remains to prove that the interval $I_i := [1 - 4m_{i+1}, 3 - 2^{i+2}]$ does not contain any integer $x \equiv 3 \pmod 4$ that is a worst NON-NADS. To this end, we first show that 3 is a $\{0, 1, x\}$-NAF for the smallest $x \in I_i$, i.e., for $x = 3 - 4m_{i+1}$. Indeed, for an odd i, it suffices to see that $3 = (101010\dots100x)_2$, where the sequence 01 is repeated $(i+1)/2$ times. This is shown by the following computation

$$(1010\dots100x)_2 = 3 - 4m_{i+1} + \sum_{j=1}^{\frac{i+1}{2}} 2^{2j+1} = \frac{17 - 2^{i+4}}{3} + 8 \cdot \frac{2^{i+1} - 1}{3} = 3.$$

We deduce that for $x_k = 4k + x$, where $k \geq 1$, we also have $(1010\dots100x_k)_2 = 3 + 4k$. Moreover, we can also show that $3 + 4k < \lfloor -\frac{x_k}{3} \rfloor$ for all $k \leq \frac{2^i - 2}{3}$ and that these x_k's correspond to all elements of the interval I_i that are congruent to 3 modulo 4. This proves that the intervals I_i's for the odd integers i's do not contain any worst NON-NADS. The case where i is even can be proved in the same way by showing that $3 = (0101\dots010x)_2$, where the sequence 01 occurred $\frac{i}{2} + 1$ times. This concludes our proof. $\square$

4 Algorithmic Considerations

We use in this section the theoretical results presented in Section 3 combined with some algorithmic methods in order to reduce the running time of the NADS search. First of all, we recall the basic algorithm (Algorithm 3) proposed by Muir and Stinson [15] and then we bring some improvements that greatly improve the performances. So, Section 4.2 takes benefit of the theoretical results of Section 3.1. Results presented in Section 3.2 are on their hand used in Section 4.3. We then give the performances of our best algorithm in Section 4.4 and show that when $x_{\max} = -10^7$, the running time of our algorithm is only 343 seconds.

```
**Algorithm 3:** Find-NADS ($x_{\max}$)

NADS ← ∅

for $i = -1$ to $i = x_{\max}$
  do { if ( Is-NADS? i )
         then NADS ← NADS ∪ $\{i\}$
       $i \leftarrow i - 4$

return NADS
```

4.1 Basic Algorithm

The algorithm Find-NADS (Algorithm 3) is the algorithm proposed by Muir and Stinson. It finds NADS from -1 to $x_{\max}$, iterating on this interval the algorithm Is-NADS? presented in Section 2.2 which aims at determining whether or not a given negative x leads to a NADS. Its performances are given in Section 4.4.

4.2 Intra-X and Inter-X Techniques

The *intra-X technique* consists of using memoization method during the execution of Is-NADS?. Memoization is an optimization technique whose basic idea is to remember function calls. A table is maintained that maps lists of argument values to previously computed return values for those arguments. When a function is called, its list of arguments is looked up in the table. If an entry is found, then the previously computed value is returned directly. Otherwise, the value is computed and then stored in the table for future use. Such a technique is well-suited for Algorithm 2 since function f_D is called many times with the same argument.

The *Inter-X technique* is an extension of the Intra-X technique using memoization during the execution of Find-NADS. Note however that the return value of f_D depends on both n and x. However, the result of $f_D(n)$ is independent of x when $n \not\equiv 3 \pmod 4$. The intuitive idea consists roughly in establishing shortcuts between n and successive iterations $f_D^k(n)$ until reaching a value congruent to 3 modulo 4. We give hereafter a formal approach, by introducing equivalence classes representing such shortcuts. Let b be the function from $\mathbb{N}$ to $\mathbb{N}$ defined by $b(n) = f_D^k(n)$ where $k \geq 0$, $f_D^k(n) \equiv 3 \pmod 4$ or $f_D^k(n) = 0$, and $\forall k'\ 0 \leq k' < k$, $f_D^{k'}(n) \not\equiv 3 \pmod 4$ and $f_D^{k'}(n) \neq 0$. We define the equivalence relation $\mathcal{R}$ such that $n\mathcal{R}n' \Leftrightarrow b(n) = b(n')$. The equivalence class of n induced by $\mathcal{R}$, denoted by $\dot{n}$ is therefore the set $\{n' \in \mathbb{N} \mid n\mathcal{R}n'\}$. The smallest element of $\dot{n}$ is called the *representative* of the class. Any element of $\dot{n}$ has a $\{0, 1, x\}$-NAF if and only if the representative of $\dot{n}$ has a $\{0, 1, x\}$-NAF. As illustration, we give on Fig. 1 some elements of the class whose representative is 7. The equivalence classes of 0 and all

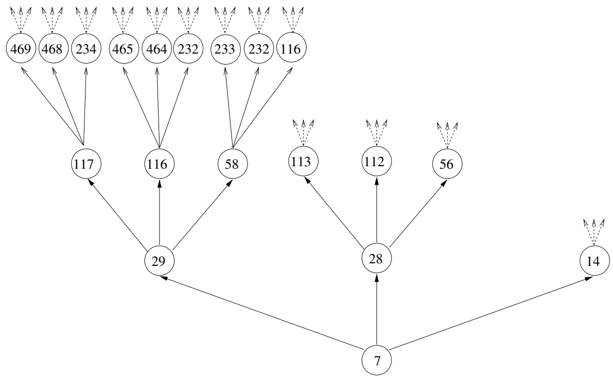

Fig. 1. Class of equivalence whose representative is 7

n such that $n \equiv 3 \pmod 4$ are therefore pre-computed and stored in a table. This pre-computation is lightweight: for every $n \equiv 3 \pmod 4$ in the interval $[0, \frac{-x}{3}]$, the inverse of the procedure f_D is recursively applied until $n > \frac{-x}{3}$. The algorithm Classes given in the appendix is the pseudo-code of this pre-computation stage. When this pre-computation is achieved, each cell of the table, indexed with n, contains the representative of $\dot{n}$. So, algorithm Is-NADS? uses the function $f_D(n)$ only when $n \equiv 3 \pmod 4$ and looks up the value in the table otherwise. Note that the table induced only a low complexity space, that is $O(|x| \log |x|)$.

4.3 Algorithm Based on Elimination of NON-NADS

We present in this section an algorithm, based on a new approach, that consists of finding all the x leading to a NADS by process of elimination of all NON-NADS. This algorithm, Elim-NON-NADS, relies on the theoretical results presented in Section 3.2. The rough idea of this algorithm is to eliminate all NON-NADS lower than a given bound $x_{\max}$ having a cycle of length t, where t varies from 1 to $\lfloor \frac{-x_{\max}}{3} \rfloor$. Indeed, x is a NON-NADS if and only if $\exists n \in \mathbb{N}, \exists t \geq 2$ such that $f_D^t(n) = n$. For instance, the cycle 3|0 yields the equation $\frac{n-x}{16} = 16$ that is $-x = 3n$. By iterating t, we can obtain all the possible values of x that reach a cycle by using a depth-first search in the the exploration tree of the different ways to construct a cycle. Using results of Section 3.2, we obtain:

$$-x = \frac{n \cdot (k_1 - 1) + k_3}{k_2} \text{ with } k_1 \geq 4,\ k_2 \geq 1,\ k_3 \geq 0,\ n \geq 3.$$

We move in the tree using the following formulas: $k_1 = 4k_1$, $k_2 = k_2$, $k_3 = k_3$, if $n \equiv 0 \pmod 4$; $k_1 = 4k_1$, $k_2 = k_2$, $k_3 = k_3 + k_1$ if $n \equiv 1 \pmod 4$; $k_1 = 2k_1$, $k_2 = k_2$, $k_3 = k_3$ if $n \equiv 2 \pmod 4$; and $k_1 = 4k_1$, $k_2 = k_2 + k_1$, $k_3 = k_3$ if $n \equiv 3 \pmod 4$. In practice, this algorithm does not allow to find all the NON-NADS when x is large due to the exponential time complexity of the tree exploration. However, it can be used to reduce the time complexity of Find-NADS by finding all NON-NADS that have cycles of length lower or equal to $t_{\max}$ such that $t_{\max}$ is small enough. Indeed, determining all NON-NADS having small cycles is much more faster with Elim-NON-NADS than with the basic Find-NADS. Consequently, finding all NON-NADS can be improved using a trade-off between Elim-NON-NADS and the basic Find-NADS. $t_{\max}$ is the *parameter* of the trade-off. As described in the appendix, Find-NADS uses Elim-NON-NADS as a sieve in a first stage in order to rough out the search process.

4.4 Experimental Results and Memory Complexity

We give in this section some experimental results in order to compare the performances of the presented algorithms. The tests were done on a standard workstation. We experimented the following algorithms whose results are given in Table 1 and represented in Fig. 2.

1. Curve A: the basic algorithm [15]; we ran the C source code that the authors gracefully provided to us.

2. Curve B: the improved basic algorithm [16]; we implemented ourself the algorithm Is-NADS? provided in [16]. We implemented then the algorithm Find-NADS using a sieve to eliminate NON-NADS characterized by Theorem 5 initially proposed in [11] and Theorem 6.
3. Curve C: our algorithm, takes benefit of our new theoretical results presented in Sections 3.1 and 3.2, and practical results described in Sections 4.2 and 4.3. It is actually a trade-off of parameter $t_{\max} = 10$ between the improved version of Find-NADS and Elim-NON-NADS. The pseudo-code of this algorithm is given in the appendix.

Note that the three implementations have been compared in a fair way (as much as possible). They have been implemented in C, compiled with the same optimization options, and executed on the same AMD Athlon$^{\text{TM}}$ XP2500+ processor. We did not try to minimize the running time of the algorithms by using some special low level functions of the language. We would like to emphasize that the memory complexity in the worse case, that is when all the cycle lengths equal

Table 1. Running time of the experimented algorithms

$x_{\max}$	Running time (seconds)		
	Basic algorithm (curve A)	Improved basic algorithm (curve B)	Our algorithm (curve C)
-10^5	7	1	<1
-10^6	655	15	3
$-3 \cdot 10^6$	1550	137	35
$-6 \cdot 10^6$	6132	435	127
$-10 \cdot 10^6$	68532	1154	343
$-13 \cdot 10^6$	–	1460	438
$-17 \cdot 10^6$	–	2454	724

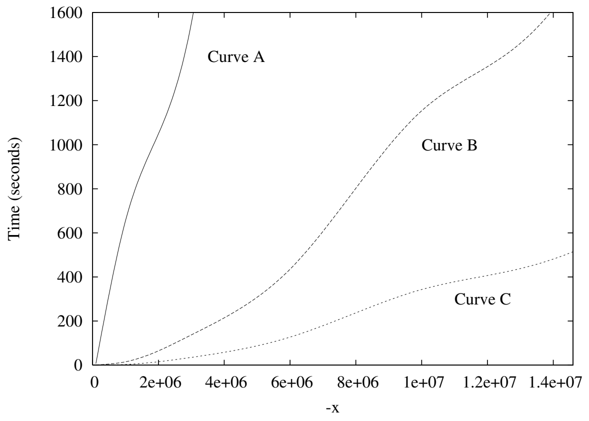

Fig. 2. Running time of the experimented algorithms

$\lfloor \frac{-x}{3} \rfloor$, is the same whatever the algorithm is. Indeed, the memory complexity is in the worse case $O(|x| \log |x|)$. Our algorithm requires however slightly more memory on average due to the precomputation steps. It fits nevertheless into a quite small memory since it requires only a few tens of megabytes of RAM.

5 Conclusion

We extended in this paper previous works mainly done by Muir and Stinson [15, 16]. Our main contribution consists of a method providing *generators* of NON-NADS infinite families and a reduction of the search domain to the interval $[0, \lfloor \frac{-x}{12} \rfloor] \cup [\lfloor \frac{-x}{7} \rfloor, \lfloor \frac{-x}{6} \rfloor]$ when x is not divided by 3 and 7. We claimed that we can still reduce it to $[0, \lfloor \frac{-x}{12} \rfloor]$. We also introduced the notion of worst NON-NADS and characterized them. From these new theoretical results, we suggested some algorithmic improvements that reduce significantly the running time of the algorithm Find-NADS. Our algorithm takes only 343 seconds when $x_{\max} = -10^7$, while the best known algorithm [16] took about 20 minutes.

References

1. A. Booth. A signed binary multiplication technique. *The Quarterly Journal Mechanics and Applied Mathematics*, 4:236–240, 1951.
2. W. Bosma. Signed bits and fast exponentiation. *Journal de théorie des nombres de Bordeaux*, 13(1):27–41, 2001.
3. E. De Win, S. Mister, B. Preneel, and M. Wiener. On the performance of signature schemes based on elliptic curves. In J. Buhler, editor, *Algorithmic Number Theory, ANTS-III*, LNCS **1423**, pp. 252–266, USA, 1998. Springer.
4. K. Fong, D. Hankerson, J. López, and A. Menezes. Field inversion and point halving revisited. *IEEE Transactions on Computers*, 53(8):1047–1059, August 2004.
5. D. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129–146, 1998.
6. J. Jedwab and C. Mitchell. Minimum weight modified signed-digit representations and fast exponentiation. *Electronics Letters*, 25(17):1171–1172, 1989.
7. M. Joye and C. Tymen. Compact encoding of non-adjacent forms with applications to elliptic curve cryptography. In K. Kim, editor, *PKC 2001*, LNCS **1992**, pp. 353–364, Korea, 2001. Springer.
8. M. Joye and S. Yen. Optimal left-to-right binary signed-digit recoding. *IEEE Transactions on Computers*, 49(7):740–748, 2000.
9. K. Koyama and Y. Tsuruoka. Speeding up elliptic cryptosystems by using a signed binary window method. In E. Brickell, editor, *CRYPTO'92*, LNCS **740**, pp. 345–357, USA, 1992. IACR, Springer.
10. L. Lalanne. Note sur quelques propositions d'arithmologie élémentaire. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 11:903–905, 1840.
11. D. Matula. Basic digit sets for radix representation. *Journal of the Association for Computing Machinery*, 29(4):1131–1143, 1982.
12. K. Okeya and T. Takagi. The Width-w NAF Method Provides Small Memory and Fast Elliptic Scalar Multiplications Secure against Side Channel Attacks. *CT-RSA*, LNCS **2612**, pp. 328–343, USA, 2003. Springer.

13. K. Okeya and T. Takagi. A More Flexible Countermeasure against Side Channel Attacks Using Window Method. *CHES 2003*, LNCS **2779**, pp. 397–410, Germany, 2003. IACR, Springer.
14. F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *RAIRO – Theoretical Informatics and Applications*, 24(6):531–543, 1990.
15. J. Muir and D. Stinson. Alternative digit sets for nonadjacent representations. In M. Matsui and R. Zuccherato, editors, *SAC 2003*, LNCS **3006**, pp. 306–319, Canada, 2003. IACR, Springer.
16. J. Muir and D. Stinson. Alternative digit sets for nonadjacent representations. Technical Report CORR 2004-09, Centre for Applied Cryptographic Research, University of Waterloo, Canada, 2004, `http://www.cacr.math.uwaterloo.ca`.
17. G. Reitwiesner. Binary arithmetic. *Advances in Computers*, 1:231–308, 1960.
18. J. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In B. Kaliski, editor, *Crypto'97*, LNCS **1294**, pp. 357–371, USA, 1997. IACR, Springer.
19. J. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, 19(2–3):195–249, 2000.

Appendix: The Final Algorithms

```
Algorithm: Find-NADS($x_{\max}$, $t_{\max}$)

$\text{NADS} \leftarrow \varnothing$
Classes($x_{\max}$)
Elim-NON-NADS($x_{\max}, t_{\max}$)

for $i = -1$ to $i = x_{\max}$
  do
    if ( Is-NADS? (i) )
      then $\text{NADS} \leftarrow \text{NADS} \cup \{i\}$
    $i \leftarrow i - 4$

return NADS
```

```
Algorithm: Is-NADS?($x$)

$N \leftarrow 3$
$T \leftarrow \varnothing$
while $N < \frac{-x}{12}$
  do
    $n \leftarrow N$
    $S \leftarrow \varnothing$

    while $n \neq 0$
      do
        if $n \in S$
          then return false
        if $n \in T$
          then break
        $S \leftarrow S \cup \{n\}$
        $T \leftarrow T \cup \{n\}$
        if $n \equiv 3 \pmod 4$
          then $n \leftarrow \frac{n-x}{4}$
        else
          then $n \leftarrow V[n]$

    $N \leftarrow N + 4$

return true
```

Fig. 3. Final algorithms

```
Algorithm: Classes($x_{\max}$)

for $i = 3$ to $i = \frac{-x_{\max}}{3}$
    do
        Classes-rec($x_{\max}, i, i$)
        $i \leftarrow i + 4$
```

```
Algorithm: Classes-rec($x_{\max}, n_{\text{seed}}, n_{\text{cur}}$)

$V[n_{\text{cur}}] \leftarrow n_{\text{seed}}$

if $n_{\text{cur}} < \frac{-x_{\max}}{3}$
    Classes-rec($x_{\max}, n_{\text{seed}}, 2 \cdot n_{\text{cur}}$)
    Classes-rec($x_{\max}, n_{\text{seed}}, 4 \cdot n_{\text{cur}}$)
    Classes-rec($x_{\max}, n_{\text{seed}}, 4 \cdot n_{\text{cur}} + 1$)
```

Fig. 4. Precomputation: the classes of equivalence

```
Algorithm: Elim-NON-NADS($x_{\max}, t_{\max}$)

Elim-NON-NADS-rec($x_{\max}, 4, 1, 0,$ "0"$, 1, t_{\max}$)
```

```
Algorithm: Elim-NON-NADS-rec($x_{\max}, k_1, k_2, k_3, e_{\text{prev}}, t_{\text{cur}}, t_{\max}$)

if $e_{\text{prev}} \neq$ "$0x$"
    for $i = 3$ to $i = \frac{-x_{\max}}{6}$
        do
            $x_{\text{cur}} = \frac{-i\cdot(k_1-1)+k_3}{k_2}$

            if ($x_{\text{cur}}$ is integer) and ($x_{\text{cur}} \equiv 3 \pmod 4$)
                then $B[x_{\text{cur}}] \leftarrow$ true

            $i \leftarrow i + 4$

if $t_{\text{cur}} \leq t_{\max}$
    if $e_{\text{prev}} \neq$ "0"
        Elim-NON-NADS-rec($x_{\max}, 4 \cdot k_1, k_2, k_3,$ "00"$, t_{\text{cur}} + 1, t_{\max}$)
        Elim-NON-NADS-rec($x_{\max}, 4 \cdot k_1, k_2, k_3 + k_1,$ "01"$, t_{\text{cur}} + 1, t_{\max}$)
        Elim-NON-NADS-rec($x_{\max}, 2 \cdot k_1, k_2, k_3,$ "0"$, t_{\text{cur}} + 1, t_{\max}$)
        Elim-NON-NADS-rec($x_{\max}, 4 \cdot k_1, k_2 + k_1, k_3,$ "$0x$"$, t_{\text{cur}} + 1, t_{\max}$)
    else
        Elim-NON-NADS-rec($x_{\max}, 4 \cdot k_1, k_2, k_3 + k_1,$ "01"$, t_{\text{cur}} + 1, t_{\max}$)
        Elim-NON-NADS-rec($x_{\max}, 4 \cdot k_1, k_2 + k_1, k_3,$ "$0x$"$, t_{\text{cur}} + 1, t_{\max}$)
```

Fig. 5. Precomputation: the sieve