New Minimal Weight Representations for Left-to-Right Window Methods[⋆]

James A. Muir[1] and Douglas R. Stinson[2⋆⋆]

[1] Department of Combinatorics and Optimization

[2] School of Computer Science

University of Waterloo
Waterloo, Ontario
Canada N2L 3G1
{jamuir,dstinson}@uwaterloo.ca

Abstract. For an integer $w \geq 2$, a radix 2 representation is called a *width-w nonadjacent form* (w-NAF, for short) if each nonzero digit is an odd integer with absolute value less than 2^{w-1}, and of any w consecutive digits, at most one is nonzero. In elliptic curve cryptography, the w-NAF *window method* is used to efficiently compute nP where n is an integer and P is an elliptic curve point. We introduce a new family of radix 2 representations which use the same digits as the w-NAF but have the advantage that they result in a window method which uses less memory. This memory savings results from the fact that these new representations can be deduced using a very simple *left-to-right* algorithm. Further, we show that like the w-NAF, these new representations have a minimal number of nonzero digits.

1 Window Methods

An operation fundamental to elliptic curve cryptography is *scalar multiplication*; that is, computing nP for an integer, n, and an elliptic curve point, P. A number of different algorithms have been proposed to perform this operation efficiently (see Ch. 3 of [4] for a recent survey). A variety of these algorithms, known as *window methods*, use the approach described in Algorithm 1.1.

For example, suppose $D = \{0, 1, 3, 5, 7\}$. Using this digit set, Algorithm 1.1 first computes and stores $P, 3P, 5P$ and $7P$. After a D-radix 2 representation of n is computed its digits are read from left to right by the "for" loop and nP is computed using doubling and addition operations (and no subtractions). One way to compute a D-radix 2 representation of n is to slide a 3-digit *window* from right to left across the $\{0, 1\}$-radix 2 representation of n (see Section 4).

Using negative digits takes advantage of the fact that subtracting an elliptic curve point can be done just as efficiently as adding it. Suppose now that $D =$

⋆ *Updated: 25 June 2004*

⋆⋆ Supported by NSERC grant RGPIN 203114-02.

$\{0, \pm 1, \pm 3\}$. Then Algorithm 1.1 computes and stores P and $3P$. As before, a D-radix 2 representation of n can computed by sliding a window of width 3 from right to left across the $\{0, 1\}$-radix 2 representation of n (see Section 4). After this representation is computed its digits are read from left to right and nP is computed using doubling, addition and subtraction operations.

```
Algorithm 1.1: RADIX-2-WINDOW-METHOD($n, P$)

fix a set of digits, $D \subset \mathbb{Z}$.
for each $d \in D$ with $d > 0$
  do $P_d \leftarrow dP$
compute and store a representation $(a_{\ell-1} \ldots a_1 a_0)_2 = n$ with $a_i \in D$.
$Q \leftarrow \infty$
for $i = \ell - 1 \ldots 0$
  do $Q \leftarrow 2Q$
     if $a_i \neq 0$
       then if $a_i > 0$
              then $Q \leftarrow Q + P_{a_i}$
              else $Q \leftarrow Q - P_{-a_i}$
return $Q$
```

Blake, Seroussi and Smart [1], Cohen, Miyaji and Ono [2] and Solinas [9] independently suggested a specialization of Algorithm 1.1 called the *width-w nonadjacent form window method* (this terminology is due to Solinas). We introduce it now.

For an integer $w \geq 2$, a radix 2 representation is called a *width-w nonadjacent form* (w-NAF, for short) if it satisfies the following conditions:

1. Each nonzero digit is an odd integer with absolute value less than 2^{w-1}.
2. Of any w consecutive digits, at most one is nonzero.

For example, the number 42 has a 3-NAF since the representation $(300\bar{3}0)_2$ (note that $\bar{1}$ denotes -1, $\bar{3}$ denotes -3, etc.) satisfies conditions 1 and 2, and

$$(300\bar{3}0)_2 = 3 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 - 3 \cdot 2^1 + 0 \cdot 2^0 = 42.$$

When $w = 2$, w-NAF coincides with the well known *nonadjacent form* [3]. Because of this, the w-NAF may be regarded as a generalization of the ordinary NAF. As with the ordinary NAF, an integer n has exactly one w-NAF, and it can be efficiently computed; hence, we refer to *the* w-NAF of n.

Let D_w be the set of w-NAF digits; that is,

$$D_w := \{0\} \cup \{d \in \mathbb{Z} : d \text{ odd}, |d| < 2^{w-1}\}.$$

If, in Algorithm 1.1, the digit set D_w is used and the representation $(a_{\ell-1} \ldots a_1 a_0)_2$ is always chosen to be a w-NAF, then this is the w-NAF window method.

One advantage of using the w-NAF of an integer is that it has a *minimal number of nonzero digits* [7]. A nonzero integer has an infinite number of D_w-radix 2 representations and any of these representation could be used in Algorithm 1.1. However, the choice of representation affects the performance of the algorithm. In the "for" loop, an addition/subtraction operation is performed for every nonzero digit of $(a_{\ell-1} \dots a_1 a_0)_2$. It is thus desirable to use a D_w-radix 2 representation of n with as few nonzero digits as possible. No other D_w-radix 2 representation of an integer has fewer nonzero digits than its w-NAF.

The w-NAF of an integer is computed by sliding a window of width w from right to left across the $\{0, 1\}$-radix 2 representation of n (see Section 4). This procedure deduces the digits of the w-NAF from right to left; however, the "for" loop of Algorithm 1.1 reads these digits from left to right. This means that the w-NAF of n must be computed and stored in its entirety before computations inside the "for" loop can begin.

This problem of opposing directions occurs in many window methods and has been lamented by both Müller [8] and Solinas [9]. If the algorithm which computes the D_w-radix 2 representation of n worked in the same direction as the "for" loop, Algorithm 1.1 could be modified so that it uses less memory. In that case, it would be unnecessary to store the representation $(a_{\ell-1} \dots a_1 a_0)_2$ since its digits could be computed inside the "for" loop as they are needed. This savings is most relevant for memory constrained devices like smartcards.

Suppose we attempt deduce the digits of the w-NAF of n from left to right. If we pursue an analogy with the right-to-left algorithm, we might try to determine the digits of the w-NAF by sliding a window of fixed width from left to right across the $\{0, 1\}$-radix 2 representation of n. Unfortunately, this method cannot succeed since it is always possible to choose an input for which such an algorithm will fail to produce a w-NAF output. There are, however, other D_w-radix 2 representations which can be computed in this manner and are just as efficient as the w-NAF.

We propose a new family of D_w-radix 2 representations and prove that, like the w-NAFs, these representations have a minimal number of nonzero digits. The digits of these representations can be deduced from left to right and thus can be used to reduce the memory requirements of Algorithm 1.1.

Joye and Yen [5] give a surprisingly elegant left-to-right algorithm for computing the digits of a $\{0, \pm 1\}$-radix 2 representation of an integer. They also prove that the representations constructed by this algorithm have a minimal number of nonzero digits. Their results apply to the digit set D_2, whereas ours apply to arbitrary D_w with $w \geq 2$. Our approach is quite different and appears to simplify some of Joye and Yen's theory.

The rest of the paper is organized as follows. In Section 2, we describe our new algorithm. In Section 3, we prove that the outputs of the algorithm have a minimal number of nonzero digits, and in Section 4, we discuss how the algorithm can be implemented efficiently. We close with a few remarks.

2 The Algorithm

In order to motivate our algorithm, we begin by describing a simple method of computing the usual $\{0,1\}$-radix 2 representation of a positive integer. This could be used if an integer were represented in some other way; for example, if we wanted to convert from radix 10 to radix 2.

Suppose we want to deduce the digits of the $\{0,1\}$-radix 2 representation of 233 from left to right. This is easily done by subtracting powers of 2. The number $n = 233$ is small enough so that we can quickly determine $2^{\lfloor \lg n \rfloor}$; this is the power of 2 closest to, but not larger than, n. Once we determine $2^{\lfloor \lg n \rfloor}$, we replace n with $n - 2^{\lfloor \lg n \rfloor}$ and then repeat these steps until we reach 0. Doing so gives us

n	$2^{\lfloor \lg n \rfloor}$
233	$128 = 2^7$
105	$64 = 2^6$
41	$32 = 2^5$
9	$8 = 2^3$
1	$1 = 2^0$.

Thus, we find that

$$233 = 2^7 + 2^6 + 2^5 + 2^3 + 2^0 = (11101001)_2.$$

We can modify this process so that it returns a $\{0,1\}$-string. We begin with a string, α, which is initially empty. In each step, we append to α a (possibly empty) run of 0's followed by a single 1. Doing so gives us

n	$2^{\lfloor \lg n \rfloor}$	α
233	$128 = 2^7$	$\alpha \parallel 1$
105	$64 = 2^6$	$\alpha \parallel 1$
41	$32 = 2^5$	$\alpha \parallel 1$
9	$8 = 2^3$	$\alpha \parallel 01$
1	$1 = 2^0$	$\alpha \parallel 001$.

Note that the symbol $\parallel$ denotes concatenation. When n reaches 0, α is equal to 11101001 and we see that $(\alpha)_2 = 233$.

For an arbitrary nonnegative integer, we can describe this process in pseudocode. Let $D = \{0, 1\}$ and define

$$\mathcal{C} := \{d \cdot 2^i : d \in D, i \in \mathbb{Z}, i \geq 0\}.$$

The set $\mathcal{C}$ contains 0 and all positive powers of 2. Here is a description of the procedure:

```
$\alpha \leftarrow \epsilon$
while $n \neq 0$
    do { $c \leftarrow$ an element in $\mathcal{C}$ closest to, but not larger than, $n$
         append digits to $\alpha$ according to the value of $c$
         $n \leftarrow n - c$
return $\alpha$
```

Note that ϵ denotes the empty string.

Consider now the digit set $D_2 = \{0, \pm 1\}$. We would like to somehow deduce the digits of a D_2-radix 2 representation of an integer from left to right. We can do this by modifying our previous procedure slightly. We first define

$$\mathcal{C}_2 := \{d \cdot 2^i : d \in D_2, i \in \mathbb{Z}, i \geq 0\}.$$

Note that $\mathcal{C}_2$ contains both the positive and negative powers of 2. Now consider the following procedure:

```
$\alpha \leftarrow \epsilon$
**while** $n \neq 0$
    **do** $c \leftarrow$ an element in $\mathcal{C}_2$ closest to $n$
       append digits to $\alpha$ according to the value of $c$
       $n \leftarrow n - c$
**return** $\alpha$
```

The only change above is that the condition "closest to, but not larger than, n" is now simply "closest to n". If we apply this procedure to $n = 233$ we have

n	c	α
233	$256 = 2^8$	$\alpha \parallel 1$
-23	$-16 = -2^4$	$\alpha \parallel 000\bar{1}$
-7	$-8 = -2^3$	$\alpha \parallel \bar{1}$
1	$1 = 2^0$	$\alpha \parallel 001$.

When n reaches 0, α is equal to $1000\bar{1}\bar{1}001$ and we see that

$$(\alpha)_2 = (1000\bar{1}\bar{1}001)_2 = 2^8 - 2^4 - 2^3 + 2^0 = 233.$$

This same example is worked in [5] and our representation is identical to the one computed by Joy and Yen.

This procedure does indeed deduce the digits of a $\{0, \pm 1\}$-radix 2 representation from left to right; however, we would like it to do so efficiently. This leads us to consider how difficult it is to identify an element in $\mathcal{C}_2$ closest to n. If we know

1. whether n is positive or negative, and
2. the $\{0, 1\}$-radix 2 representation of $|n|$,

then this is not difficult at all. For example, 233 is positive and $|233| = (11101001)_2$. The most significant digit of this representation tells us that

$$2^7 \leq 233 < 2^8$$

and the two most significant digits together tell us that 233 is as close as, or closer, to 2^8 than 2^7. If this procedure is implemented in digital hardware, then, most likely, facts 1. and 2. are already known for a given input, n. Moreover, if we have the $\{0, 1\}$-radix 2 representation of $|n|$ then from it we can determine the $\{0,1\}$-radix 2 representation of $|n-c|$. For example, $\left|233-2^8\right| = 23$ and the $\{0,1\}$-radix 2 representation of 23 is deduced by simply xor-ing each digit, except the least significant, of the $\{0,1\}$-radix 2 representation of 233 with 1:

$$\begin{aligned} 233 &= (11101001)_2 \\ \left|233-2^8\right| = 23 &= (00010111)_2. \end{aligned}$$

In this case, $|c| > |n|$; if $|c| \leq |n|$ then the representation of $|n-c|$ is deduced by just changing the most significant digit of the $\{0,1\}$-radix 2 representation of $|n|$ to 0.

In the general case, we would like to construct D_w-radix 2 representations from left to right for arbitrary $w \geq 2$. Here is a procedure which does so:

```
Algorithm 2.1: MSF_w(n)

  comment: $w \geq 2,\ D_w = \{0\} \cup \{d \in \mathbb{Z} : d \text{ odd}, |d| < 2^{w-1}\},$ and
           $\mathcal{C}_w = \{d \cdot 2^i : d \in D_w, i \in \mathbb{Z}, i \geq 0\}$

  $\alpha \leftarrow \epsilon$
  while $n \neq 0$
       { $c \leftarrow$ an element in $\mathcal{C}_w$ closest to $n$
    do { append digits to $\alpha$ according to the value of $c$
       { $n \leftarrow n - c$
  return $\alpha$
```

As in the case for $w = 2$, this procedure can be implemented efficiently by examining the w most significant digits of the $\{0,1\}$-radix 2 representation of $|n|$. (A description of how digits are appended to α will be provided a bit later.) We call this procedure $\mathrm{MSF}_w(n)$.

We have given this procedure the title "Algorithm". To justify this we must show that $\mathrm{MSF}_w(n)$ terminates for all $n \in \mathbb{Z}$. If $n = 0$, then $\mathrm{MSF}_w(n)$ clearly terminates, so we need only consider $n \neq 0$. To finish the argument we need a Lemma.

Lemma 1. *Let n be a nonzero integer. If c is an element in $\mathcal{C}_w$ closest to n, then*

$$|n-c| \leq \frac{2^{\lfloor \lg |n| \rfloor}}{2^{w-1}}.$$

Proof. We will assume $n > 0$; the proof for $n < 0$ is similar. First, suppose $n < 2^{w-1}$. If n is odd then $n \in D_w$ and hence $n = n \cdot 2^0 \in \mathcal{C}_w$. If n is even then $n = d \cdot 2^i$ for some $i < w-1$ and $d \in D_w$ and hence $n = d \cdot 2^i \in \mathcal{C}_w$. Thus, if $n < 2^{w-1}$ and c is closest to n then $c = n$. So, $n - c = 0$ and the inequality is valid.

Consider the closed interval $[2^{w-2}, 2^{w-1}]$. This interval contains exactly $2^{w-2}+1$ integers and, as we have just seen, each one is in $\mathcal{C}_w$. More generally, for $i \geq w-2$, the interval $[2^i, 2^{i+1}]$ contains exactly $2^{w-2}+1$ elements of $\mathcal{C}_w$. Moreover, these elements partition the interval into 2^{w-2} subintervals of equal length.

Now, suppose $n \geq 2^{w-1}$. The integer n lies in the interval $[2^{\lfloor \lg n \rfloor}, 2^{\lfloor \lg n \rfloor + 1}]$ which has length $2^{\lfloor \lg n \rfloor}$. Since $\lfloor \lg n \rfloor \geq w - 1$, this interval contains exactly $2^{w-2} + 1$ elements of $\mathcal{C}_w$. These elements partition this interval into subintervals of length $2^{\lfloor \lg n \rfloor}/2^{w-2}$. The integer n lies in a subinterval with endpoint c. Since both endpoints of this subinterval are in $\mathcal{C}_w$, we have

$$|n - c| \leq 1/2 \cdot (2^{\lfloor \lg n \rfloor}/2^{w-2}) = 2^{\lfloor \lg n \rfloor}/2^{w-1}.$$

Since n is positive, we have that $|n| = n$, and this gives us the desired inequality. □

To show that $\mathrm{MSF}_w(n)$ terminates for $n \neq 0$, it suffices to show that $|n| > |n - c|$. Suppose to the contrary that $|n| \leq |n - c|$. Then

$$\begin{aligned} |n| &\leq |n - c| \\ \Longrightarrow\ |n| &\leq 2^{\lfloor \lg |n| \rfloor}/2^{w-1} \qquad \text{(by Lemma 1)} \\ \Longrightarrow\ 2^{\lfloor \lg |n| \rfloor} &\leq 2^{\lfloor \lg |n| \rfloor}/2^{w-1} \\ \Longrightarrow\ 1 &\leq 1/2^{w-1} \\ \Longrightarrow\ w &\leq 1. \end{aligned}$$

However, this is a contradiction because $w \geq 2$. So, the sequence formed by taking the absolute value of the variable n during the execution of $\mathrm{MSF}_w(n)$ is strictly decreasing. Thus, the variable n must reach 0, and so $\mathrm{MSF}_w(n)$ terminates for all $n \in \mathbb{Z}$.

The string α returned by $\mathrm{MSF}_w(n)$ has been defined somewhat informally. We present a more rigorous definition based on the values that the variable c assumes during the execution of $\mathrm{MSF}_w(n)$. For an input, n, we define $\alpha = \ldots a_2 a_1 a_0$ to be the string such that

$$a_i := \begin{cases} d & \text{if } c \text{ assumes the value } d \cdot 2^i \text{ at some point in the algorithm,} \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Clearly, each $a_i \in D_w$, and so α is a D_w-string. There is, however, one possible problem with this definition. Suppose c assumes the two distinct values $d_0 \cdot 2^i$ and $d_1 \cdot 2^i$ which share the same power of 2. In that case, the value of a_i is undefined. Fortunately, this problem never occurs.

Lemma 2. *Let c_0, c_1 and n be nonzero integers such that c_0 is an element in $\mathcal{C}_w$ closest to n and c_1 is an element in $\mathcal{C}_w$ closest to $n - c_0$. If $c_0 = d_0 2^{i_0}$ and $c_1 = d_1 2^{i_1}$ with $d_0, d_1 \in D_w$, then $i_0 > i_1$.*

Proof. We begin by bounding $|c_0|$. Since $|n|$ lies in the interval $[2^{\lfloor \lg |n| \rfloor}, 2^{\lfloor \lg |n| \rfloor + 1}]$ and both endpoints of this interval are in $\mathcal{C}_w$ we have

$$\begin{aligned} 2^{\lfloor \lg |n| \rfloor} &\leq |c_0| \leq 2^{\lfloor \lg |n| \rfloor + 1} \\ \Longrightarrow 2^{\lfloor \lg |n| \rfloor} &\leq |d_0|\, 2^{i_0} \leq 2^{\lfloor \lg |n| \rfloor + 1}. \end{aligned}$$

Thus,

$$\begin{aligned} |d_0| = 1 &\implies i_0 = \lfloor \lg |n| \rfloor \text{ or } \lfloor \lg |n| \rfloor + 1, \text{ and} \\ |d_0| > 1 &\implies i_0 = \lfloor \lg |n| \rfloor - \lfloor \lg |d_0| \rfloor . \end{aligned}$$

Since $d_0 \in D_w$, $|d_0| < 2^{w-1}$ and thus $\lfloor \lg |d_0| \rfloor \in \{0, 1, \ldots w-2\}$. From these implications we can conclude that

$$i_0 \in \{\lfloor \lg |n| \rfloor - w + 2, \ldots, \lfloor \lg |n| \rfloor + 1\}.$$

In the same way, we can deduce that

$$i_1 \in \{\lfloor \lg |n - c_0| \rfloor - w + 2, \ldots, \lfloor \lg |n - c_0| \rfloor + 1\}.$$

Now, by Lemma 1, we have

$$\begin{aligned} &|n - c_0| \leq 2^{\lfloor \lg |n| \rfloor - w + 1} \\ \implies &2^{\lfloor \lg |n - c_0| \rfloor} \leq 2^{\lfloor \lg |n| \rfloor - w + 1} \\ \implies &\lfloor \lg |n - c_0| \rfloor \leq \lfloor \lg |n| \rfloor - w + 1 \\ \implies &\lfloor \lg |n - c_0| \rfloor + 1 \leq \lfloor \lg |n| \rfloor - w + 2. \end{aligned}$$

Thus, the largest possible value of i_1 is less than or equal to the smallest possible value of i_0. Hence $i_1 \leq i_0$.

To see that $i_1 \neq i_0$, first note that

$$|n| > |n - c_0| > |(n - c_0) - c_1| ,$$

and so

$$|n - c_0| > |n - (c_0 + c_1)| .$$

Thus, the integer $c_0 + c_1$ is closer to n than c_0 is. If $i_1 = i_0$, then

$$c_0 + c_1 = (d_0 + d_1)2^{i_0}.$$

Since d_0 and d_1 are both odd, $d_0 + d_1 = d \cdot 2^i$ for some $d \in D_w \setminus \{0\}$ and $i \geq 1$. Thus, $c_0 + c_1 \in \mathcal{C}_w$. However, this contradicts the fact that c_0 is closest to n. So it must be that $i_1 \neq i_0$, and hence $i_1 < i_0$, as required. □

By Lemma 2, the string α is well defined. Moreover, as we saw in our earlier examples, Lemma 2 tells us that α can be constructed using operations of the form

$$\alpha \leftarrow \alpha \parallel 0^t d \quad \text{where} \quad t \geq 0,\ d \in D_w,\ d > 0.$$

Actually, we should be a bit more precise here. If n is odd, then α can be constructed using only operations like the one above; however, if n is even, then α will need to have a run of zeros appended to it before it is returned.

From the definition given in (1) we can now show that the string returned by $\mathrm{MSF}_w(n)$ is in fact a D_w-radix 2 representation of n (i.e., the algorithm is correct). Let S be the set of values that the variable c assumes during the execution of $\mathrm{MSF}_w(n)$. For $\alpha = \ldots a_1 a_0$ we have

$$(\alpha)_2 = \sum_{i \geq 0} a_i 2^i = \sum_{\substack{i \\ a_i \neq 0}} a_i 2^i = \sum_{c \in S} c = n.$$

The representation returned by $\mathrm{MSF}_w(n)$ for a given value of n is not necessarily unique. For example, when $w = 3$, $D_3 = \{0, \pm 1, \pm 3\}$ and for $n = 5$ we see that both $4 = 2^2$ and $6 = 3 \cdot 2^1$ are elements in $\mathcal{C}_3$ closest to 5. Thus, $\mathrm{MSF}_3(5)$ will return one of the representations

$$(101)_2 \quad \text{or} \quad (3\bar{1})_2.$$

From the description of Algorithm 2.1, it is apparent that $\mathrm{MSF}_w(n)$ will have more than one possible output only when some value of the variable n has more than one closest element in $\mathcal{C}_w$. This occurs only when a value of the variable n is the midpoint between neighbouring elements of $\mathcal{C}_w$.

We argue that there are at most two distinct outputs of $\mathrm{MSF}_w(n)$ for any $n \in \mathbb{Z}$. Imagine a list of outputs of $\mathrm{MSF}_w(n)$. Let i_0 be the largest value of i such that two outputs differ at digit i. If i_0 does not exist then all the outputs are the same; otherwise, let $\alpha = \ldots a_2 a_1 a_0$ and $\beta = \ldots b_2 b_1 b_0$ be two outputs with $a_{i_0} \neq b_{i_0}$. Let

$$n' = n - \sum_{i > i_0} a_i 2^i = n - \sum_{i > i_0} b_i 2^i.$$

One of a_{i_0} and b_{i_0} is nonzero. We assume, without loss of generality, that $a_{i_0} \neq 0$. Let $c_0 = a_{i_0} 2^{i_0}$. The value c_0 is an element in $\mathcal{C}_w$ is closest to n'. Since $a_{i_0} \neq b_{i_0}$ there must be another value, say c_1, closest to n', and this value must be encoded as the most significant nonzero digit of the string $b_{i_0} \ldots b_1 b_0$. Since both c_0 and c_1 are closest to n', n' must be the midpoint between c_0 and c_1. Thus, $|n' - c_0|$ is as large as possible, so by Lemma 1 we have

$$|n' - c_0| = |n' - c_1| = 2^{\lfloor \lg |n'| \rfloor - w + 1}.$$

Let $t = \lfloor \lg |n'| \rfloor - w + 1$. Since $n' - c_0 = \pm 2^t$, $n' - c_0 \in \mathcal{C}_w$ and $n' - c_0$ is the unique element in $\mathcal{C}_w$ closest to $n' - c_0$. Similarly, $n' - c_1$ is the unique element in $\mathcal{C}_w$ closest to $n' - c_1$. Thus, the least significant nonzero digits of α and β correspond to the values $n' - c_0$ and $n' - c_1$; that is, the least significant nonzero digits of α and β are a_t and b_t where one of a_t, b_t is 1 and the other -1 (note that the example above has this property). Thus, there just two kinds of outputs: ones that encode c_0 and ones that encode c_1.

From the preceding discussion, we can derive the following Lemma:

Lemma 3. *Let α and β be two outputs of $MSF_w(n)$. Then α and β have the same number of nonzero digits.*

Proof. If $\alpha = \beta$ then there is nothing to prove, so we can assume $\alpha \neq \beta$. Let

$$\alpha = \ldots a_{i_0} \ldots a_1 a_0 \quad \text{and} \quad \beta = \ldots b_{i_0} \ldots b_1 b_0,$$

where i_0 is the largest value of i such that $a_i \neq b_i$. From our discussion above, the strings $a_{i_0} \ldots a_1 a_0$ and $b_{i_0} \ldots b_1 b_0$ each contain exactly two nonzero digits. Thus α and β have the same number of nonzero digits. □

It is possible to implement Algorithm 2.1 in such a way that it returns a unique representation for every $n \in \mathbb{Z}$. For example, if c_0 and c_1 are both closest to n then we might resolve this ambiguity by always choosing the one of larger absolute value. Because of Lemma 3, we know that, however Algorithm 2.1 is implemented, all outputs will have the same number of nonzero digits (for a given input). In fact, any representation of n constructed by Algorithm 2.1 will have a *minimal number of nonzero digits*. Proving this fact is the topic of the next section.

3 Minimality

If α is a string of digits, we denote the number of nonzero digits in α by $\mathsf{wt}(\alpha)$. We will refer to $\mathsf{wt}(\alpha)$ as the *weight* of the string α. The set of all strings composed of digits in D_w is denoted by ${D_w}^*$. For an integer n, we define

$$\mathsf{wt}^*(n) := \min\{\mathsf{wt}(\alpha) : \alpha \in {D_w}^*, (\alpha)_2 = n\}.$$

So, $\mathsf{wt}^*(n)$ is the minimum number of nonzero digits required to represent n using a D_w-radix 2 representation. If $\alpha \in {D_w}^*$ and $(\alpha)_2 = n$ then it must be that $\mathsf{wt}(\alpha) \geq \mathsf{wt}^*(n)$; if $\mathsf{wt}(\alpha) = \mathsf{wt}^*(n)$ we say that α has *minimal weight.*

In this section, we will prove the following Theorem:

Theorem 4. *Let $w \geq 2$ be an integer. For any $n \in \mathbb{Z}$, the representation returned by $MSF_w(n)$ has a minimal number of nonzero digits.*

It will be convenient to let $\mathrm{MSF}_w(n)$ denote a string returned by the algorithm on input n. To prove Theorem 4 we will show that for any $n \in \mathbb{Z}$, $\mathsf{wt}(\mathrm{MSF}_w(n)) = \mathsf{wt}^*(n)$. In doing so, we will make use of a number of short Lemmas concerning the functions $\mathsf{wt}^*(n)$ and $\mathsf{wt}(\mathrm{MSF}_w(n))$.

Lemma 5. *If n is even then* $\mathsf{wt}^*(n) = \mathsf{wt}^*(n/2)$.

Proof. Let $(\ldots a_2 a_1 a_0)_2$ be a minimal weight representation of n. Since n is even, $a_0 = 0$ and so $(\ldots a_2 a_1)_2 = n/2$. Thus, $\mathsf{wt}^*(n/2) \leq \mathsf{wt}^*(n)$. Let $(\ldots b_2 b_1 b_0)_2$ be a minimal weight representation of $n/2$. Then $(\ldots b_2 b_1 b_0 0)_2 = n$ and so $\mathsf{wt}^*(n) \leq \mathsf{wt}^*(n/2)$. □

For any $n \in \mathbb{Z}$, there exist a unique pair of integers, q and r, such that

$$n = q \cdot 2^w + r \quad \text{where} \quad -2^{w-1} < r \leq 2^{w-1}.$$

We will denote this value of r by "n mods 2^w". For example, if $w = 3$ then 13 mods $2^3 = -3$. Note that if n is odd then so is n mods 2^w. As well, when

$n > 0$ it must be that $q \geq 0$, and similarly, when $n < 0$, $q \leq 0$. So, for $n \neq 0$, we have $q/n \geq 0$.

If we write $n = q \cdot 2^w + r$ with $r = n \mod\text{s}\ 2^w$ then $q \cdot 2^w$ is a *multiple of* 2^w *closest* to n. We will make use of this fact later on.

The w-NAF of an integer has minimal weight [7]. If n is odd then the least significant digit of its w-NAF is equal to $n \mod\text{s}\ 2^w$. From this fact, we can deduce the following Lemma:

Lemma 6. *If n is odd and $r = n \mod\text{s}\ 2^w$, then $\mathsf{wt}^*(n) = 1 + \mathsf{wt}^*((n-r)/2)$.*

Lemma 6 can proved in the same way as Lemma 5.

To show that $\mathsf{wt}(\mathrm{MSF}_w(n)) = \mathsf{wt}^*(n)$, we will argue by induction on $|n|$. For n odd, it is thus useful to establish that $|(n-r)/2| < |n|$.

Lemma 7. *Let n be an odd integer and let $r = n \mod\text{s}\ 2^w$. Then*

$$\left|\frac{n-r}{2}\right| < |n| .$$

Proof. If $|n| < 2^{w-1}$ then $n \mod\text{s}\ 2^w = n$. Thus,

$$\left|\frac{n-r}{2}\right| = \left|\frac{n-n}{2}\right| = 0 < |n| .$$

So we can assume that $|n| \geq 2^{w-1}$. Let q be the integer such that $n = q \cdot 2^w + r$. Since n is nonzero, we have

$$\left|\frac{n-r}{2}\right| < |n| \iff \left|\frac{n-r}{n}\right| < 2 \iff \left|\frac{q}{n}\right| 2^w < 2.$$

Suppose to the contrary that $\left|\frac{q}{n}\right| 2^w \geq 2$. The integers q and n have the same sign, thus

$$\begin{aligned} \left|\frac{q}{n}\right| 2^w \geq 2 &\implies \frac{q}{n}2^w \geq 2 \\ &\implies \frac{q}{n}2^w + \frac{r}{n} \geq 2 + \frac{r}{n} \\ &\implies 1 \geq 2 + \frac{r}{n}. \end{aligned} \tag{2}$$

Because $r = n \mod\text{s}\ 2^w$, we know that $-2^{w-1} < r \leq 2^{w-1}$. However, n is odd, so r is odd and we have the slightly tighter bound $-2^{w-1} < r < 2^{w-1}$. Since $|n| \geq 2^{w-1}$ we have $1 \geq 2^{w-1}/|n|$, thus

$$\begin{aligned} -2^{w-1} < r < 2^{w-1} &\implies \frac{-2^{w-1}}{|n|} < \frac{r}{|n|} < \frac{2^{w-1}}{|n|} \\ &\implies -1 < \frac{r}{|n|} < 1 \\ &\implies -1 < \frac{r}{n} < 1. \end{aligned}$$

Since $r/n > -1$, if we continue from (2) we arrive at the contradiction $1 > 1$. Thus, $\left|\frac{q}{n}\right| 2^w < 2$ and this is equivalent to the desired result. □

We now give two Lemmas which involve the function $\mathsf{wt}(\mathrm{MSF}_w(n))$.

Lemma 8. *If n is an even integer then* $\mathsf{wt}(\mathrm{MSF}_w(n)) = \mathsf{wt}(\mathrm{MSF}_w(n/2))$.

Proof. If $n = 0$ then the result is clearly true, so we can assume $n \neq 0$. Let $\alpha = a_{\ell-1} \dots a_2 a_1 a_0$ be an output of $\mathrm{MSF}_w(n)$ with $a_{\ell-1} \neq 0$. Since n is even and $n = (\alpha)_2$ it must be that $a_0 = 0$. Thus, the strings α and $\alpha' = a_{\ell-1} \dots a_2 a_1$ have the same weight. We show α' is an output of $\mathrm{MSF}_w(n/2)$, and then the result follows from Lemma 3.

Let $c = a_{\ell-1} 2^{\ell-1}$; c is an element in $\mathcal{C}_w$ closest to n. Since $a_0 = 0$ and $a_{\ell-1} \neq 0$ it must be that $\ell - 1 \geq 1$, and so c is even. Thus, $c/2 \in \mathcal{C}_w$. Now,

$$c \text{ is closest to } n \implies c/2 \text{ is closest to } n/2,$$

so there is an output of $\mathrm{MSF}_w(n/2)$ where the most significant nonzero digit encodes $c/2 = a_{\ell-1} 2^{\ell-2}$. By repeating this argument, we see that $\alpha' = a_{\ell-1} \dots a_2 a_1$ is indeed an output of $\mathrm{MSF}_w(n/2)$. This proves the result. □

Lemma 9. *If c is an element of $\mathcal{C}_w$ closest to n, then* $\mathsf{wt}(\mathrm{MSF}_w(n)) = 1 + \mathsf{wt}(\mathrm{MSF}_w(n - c))$.

Lemma 9 follows from the description of Algorithm 2.1.

Now we have everything we need to prove our main result.

Proof (of Theorem 4). We show that for any $n \in \mathbb{Z}$,

$$\mathsf{wt}(\mathrm{MSF}_w(n)) = \mathsf{wt}^*(n)\,. \tag{3}$$

When $n = 0$, $\mathrm{MSF}_w(n)$ returns the empty string; thus

$$\mathsf{wt}(\mathrm{MSF}_w(0)) = 0 = \mathsf{wt}^*(0)\,. \tag{4}$$

Also, if n is even then from Lemmas 5 and 8 we have

$$\mathsf{wt}(\mathrm{MSF}_w(n)) = \mathsf{wt}^*(n) \iff \mathsf{wt}(\mathrm{MSF}_w(n/2)) = \mathsf{wt}^*(n/2)\,. \tag{5}$$

Thus, if we can show that (3) holds for all n with $|n| \geq 1$ and n odd, then by (4) and (5), it holds for all n.

Let n be an odd nonzero integer. We argue by induction on $|n|$. For our base cases, we consider the values of n that satisfy $1 \leq |n| < 2^{2w-1}$. We deal with this interval in two parts.

First, suppose $1 \leq |n| < 2^{w-1}$. Then $n \in D_w$ (because n is odd) and thus $\mathsf{wt}(\mathrm{MSF}_w(n)) = 1$. Any odd integer n has $\mathsf{wt}^*(n) \geq 1$, thus we see that

$$\mathsf{wt}(\mathrm{MSF}_w(n)) = 1 = \mathsf{wt}^*(n)\,.$$

Next, suppose $2^{w-1} \leq |n| < 2^{2w-1}$. Note that $\lfloor \lg |n| \rfloor \leq 2w - 2$. Let c be an element in $\mathcal{C}_w$ closest to n. Note that c must be even since $|n| \geq 2^{w-1}$. By Lemma 1, we have

$$|n - c| \leq 2^{\lfloor \lg |n| \rfloor - w + 1}. \tag{6}$$

However,

$$\lfloor \lg |n| \rfloor \leq 2w - 2 \implies \lfloor \lg |n| \rfloor - w + 1 \leq w - 1,$$

and so

$$|n - c| \leq 2^{w-1}.$$

Since n is odd and c is even, $n - c$ is odd and thus, $n - c \in D_w$. So Algorithm 2.1 uses just two elements of $\mathcal{C}_w$ to represent n (namely, c and $n - c$); thus $\mathsf{wt}(\mathrm{MSF}_w(n)) = 2$. Any odd integer n with $|n| > 2^{w-1}$ (i.e., $n \notin D_w$) has $\mathsf{wt}^*(n) \geq 2$, and from this we see that

$$\mathsf{wt}(\mathrm{MSF}_w(n)) = 2 = \mathsf{wt}^*(n)\,.$$

With our base cases established, we now consider n odd with $|n| \geq 2^{2w-1}$. Note that $\lfloor \lg |n| \rfloor \geq 2w - 1$. Let c be an element in $\mathcal{C}_w$ closest to n and let $r = n \text{ mods } 2^w$. We claim that c is also closest to $n - r$. To see this, first note that n lies in one of the intervals

$$[2^{\lfloor \lg |n| \rfloor}, 2^{\lfloor \lg |n| \rfloor + 1}] \quad \text{or} \quad [-2^{\lfloor \lg |n| \rfloor}, -2^{\lfloor \lg |n| \rfloor + 1}].$$

From the proof of Lemma 2, we know that all elements of $\mathcal{C}_w$ in these intervals have the form $d \cdot 2^i$ with $d \in D_w$ and

$$i \in \{\lfloor \lg |n| \rfloor - w + 2, \ldots, \lfloor \lg |n| \rfloor, \lfloor \lg |n| \rfloor + 1\}.$$

Thus,

$$i \geq \lfloor \lg |n| \rfloor - w + 2 \geq 2w - 1 - w + 2 = w + 1,$$

and so 2^{w+1} divides c. There are two neighbouring elements of $\mathcal{C}_w$, say c_0 and c_1, such that $n \in [c_0, c_1]$. Let m be the midpoint of $[c_0, c_1]$. We have

$$\begin{aligned} 2^{w+1} | c_0 \text{ and } 2^{w+1} | c_1 &\implies 2^{w+1} | (c_0 + c_1) \\ &\implies 2^w | \frac{c_0 + c_1}{2} \\ &\implies 2^w | m. \end{aligned}$$

So c_0, c_1 and m are all multiples of 2^w. One of c_0 or c_1 is equal to c. If $c = c_0$, then $n \in [c, m]$; whereas, if $c = c_1$, then $n \in [m, c]$. In either case, it can be shown that $n - r$ is an element in the same closed interval (this follows because $n - r$ is the multiple of 2^w closest to n). Thus, we see that c is closest to $n - r$. Further, since both c and $n - r$ are even, we have that

$$c/2 \text{ is closest to } (n - r)/2. \tag{7}$$

As an example, suppose that $w = 4$ and $n = 349$. The elements of $\mathcal{C}_4$ between 256 and 512 are $256, 320, 384, 448, 512$. Therefore $c = 320$. On the other hand, $r = 349 \text{ mods } 16 = -3$, so $n - r = 352$ and $(n - r)/2 = 176$. Equation (7) asserts that $c/2 = 160$ is an element of $\mathcal{C}_4$ closest to 176. This is indeed true, because the elements of $\mathcal{C}_4$ between 128 and 256 are $128, 160, 192, 224, 256$.

Now we are ready to finish the proof. Notice that, because $2^w | c$, we have

$$n - c \ \text{mods}\ 2^w = n \ \text{mods}\ 2^w = r. \tag{8}$$

By induction, we have that $\mathsf{wt}(\mathrm{MSF}_w(n')) = \mathsf{wt}^*(n')$ for all n' with $|n'| < |n|$. Using this and our Lemmas, we find that

$$\begin{aligned}
\mathsf{wt}(\mathrm{MSF}_w(n)) &= 1 + \mathsf{wt}(\mathrm{MSF}_w(n-c)) \qquad \text{(by Lemma 9)} \\
&= 1 + \mathsf{wt}^*(n-c) \qquad \text{(by induction)} \\
&= 1 + 1 + \mathsf{wt}^*\left(\frac{(n-c)-r}{2}\right) \qquad \text{(by (8) and Lemma 6)} \\
&= 1 + 1 + \mathsf{wt}\left(\mathrm{MSF}_w\left(\frac{(n-c)-r}{2}\right)\right) \qquad \text{(by induction)} \\
&= 1 + 1 + \mathsf{wt}\left(\mathrm{MSF}_w\left(\frac{n-r}{2} - \frac{c}{2}\right)\right) \\
&= 1 + \mathsf{wt}\left(\mathrm{MSF}_w\left(\frac{n-r}{2}\right)\right) \qquad \text{(by (7) and Lemma 9)} \\
&= 1 + \mathsf{wt}^*\left(\frac{n-r}{2}\right) \qquad \text{(by induction)} \\
&= \mathsf{wt}^*(n) \qquad \text{(by Lemma 6).}
\end{aligned}$$

Each of the inductive steps above is justified by either Lemma 7 or the fact that $|n - c| < |n|$. This concludes the proof. □

4 Implementations

We first review two known right-to-left sliding window methods for constructing D-radix 2 representations. After this, we give an example of how Algorithm 2.1 can implemented using a left-to-right sliding window method.

4.1 Right-to-Left Representations

Suppose we want to deduce a radix 2 representation of the integer 379 using the digits $D = \{0, 1, 3, 5, 7\}$. If we know the $\{0, 1\}$-radix 2 representation of 379 then this is easily done. Consider the following table which maps 3-digit strings to 3-digit strings:

β	β'
001	001
011	003
101	005
111	007

In each row of the table, the string β' corresponds to $(\beta)_2 \mod 2^3$. Using the $\{0, 1\}$-radix 2 representation of 379 and our table, we perform the following operations:

$$\begin{array}{r|r} 379 = (101111\underline{011})_2 & 003 \\ (101\underline{111}011)_2 & 007003 \\ (\underline{101}111011)_2 & 005007003 \end{array}$$

This sequence of operations can be described by considering a 3-digit *window* which slides right to left across the $\{0,1\}$-radix 2 representation. Anytime the digits in the window match one of the entries in the left hand column of our table we output the corresponding 3-digit string and then advance the window 3 digits to the left. Otherwise, we output a single 0 and advance the window 1 digit to the left.

This process can also be described in terms of integer operations, as shown in Algorithm 4.1. Note that the integer w denotes the width of the window.

```
Algorithm 4.1: WIDTH-$w$-REPRESENTATION($n$)

$\alpha \leftarrow \epsilon$
while $n \neq 0$
  do  if $n \bmod 2 = 1$
        then $a_i \leftarrow n \bmod 2^w$
        else $a_i \leftarrow 0$
      $\alpha \leftarrow a_i \parallel \alpha$
      $n \leftarrow (n - a_i)/2$
return $\alpha = \ldots a_2 a_1 a_0$
```

Suppose that we now want to use the digits $D_3 = \{0, \pm 1, \pm 3\}$. We can construct a D_3-radix 2 representation of 379 using a similar process. We begin with the following table:

β	c	β'	c'
001	0	001	0
011	0	003	0
101	0	$00\bar{3}$	1
111	0	$00\bar{1}$	1
000	1	001	0
010	1	003	0
100	1	$00\bar{3}$	1
110	1	$00\bar{1}$	1

This table describes a map, $(\beta, c) \mapsto (\beta', c')$, between ordered pairs. The ordered pairs consist of a 3-digit string and a *carry*, c. Notice that for each row of the table, the string β' corresponds to $((\beta)_2 + c)$ mods 2^3. After initializing the carry to 0, we can apply these transformations using a sliding 3-digit window:

$$\begin{array}{r|r}
379 = (0101111\underline{01\overset{0}{1}})_2 & 003 \\
(0101\underline{11\overset{0}{1}}011)_2 & 00\bar{1}003 \\
(0\underline{10\overset{1}{1}}111011)_2 & 000\bar{1}003 \\
(\underline{01\overset{1}{0}}1111011)_2 & 003000\bar{1}003
\end{array}$$

Each time the contents of the window and the value of the carry match an entry in the left hand column of our table we output the corresponding 3-digit string, update the carry and then advance the window 3 digits to the left. Otherwise, we output a single 0, leave the carry unchanged and advance the window 1 digit to the left.

This process constructs an integer's 3-NAF, and it does so using only a look-up table. If we allow simple bit operations, like xor, the number of rows in the table can be halved. This process can also be described in terms of integer operations, as in Algorithm 4.2.

Algorithm 4.2: $\mathrm{NAF}_w(n)$

```
$\alpha \leftarrow \epsilon$
**while** $n \neq 0$
  **do** {
      **if** $n \bmod 2 = 1$
        **then** $a_i \leftarrow n$ mods $2^w$
        **else** $a_i \leftarrow 0$
      $\alpha \leftarrow a_i \parallel \alpha$
      $n \leftarrow (n - a_i)/2$
  }
**return** $\alpha = \ldots a_2 a_1 a_0$
```

4.2 Left-to-Right Representations

Using Algorithm 2.1, we can deduce the digits of a D_3-radix 2 representation of 379 from left to right. The simple description of Algorithm 2.1 given in Section 2 allowed us to prove a number of properties about the algorithm and its outputs. However, when faced with the task of actually implementing the algorithm in software or hardware, a description based on a look-up table may be more convenient.

Consider the following table

β	s	β'	s'
0100	0	010	0
0101	0	003	1
0110	0	003	0
0111	0	100	1
1000	1	$\bar{1}00$	0
1001	1	$00\bar{3}$	1
1010	1	$00\bar{3}$	0
1011	1	$0\bar{1}0$	1

As in our previous example, this table describes a map between ordered pairs. The relation between the strings β and β' is based on selecting an element in $\mathcal{C}_3$ closest to a certain integer. Consider, for example, the first four rows of the table. If we add a leading zero to the $\{0, 1\}$-radix 2 representation of a positive integer, then the four digit prefix of this representation must equal one of the four values of β. Note that we can always extract a four digit prefix by taking zeros from the right of the radix point when necessary. If this prefix is equal to

$$(0101\ldots)_2 = 0 \cdot 2^{\ell} + 1 \cdot 2^{\ell-1} + 0 \cdot 2^{\ell-2} + 1 \cdot 2^{\ell-3} + \cdots$$

then the closest element in $\mathcal{C}_3$ is

$$(0030\ldots0)_2 = 0 \cdot 2^{\ell} + 0 \cdot 2^{\ell-1} + 3 \cdot 2^{\ell-2}.$$

Hence the prefix 0101 is mapped to the string 003. However, if we take a positive integer with prefix 0101 and subtract a closest element in $\mathcal{C}_3$, we end up with a negative integer. This means that we somehow need to keep track of the sign of an integer. The variable s serves this purpose.

The last four rows of the table correspond to negative integers. Each value of β in these rows is read using a *ones' complement* notation (cf. [6], page 203), and this is indicated by the value of the variable s. For example, consider the fifth row of the table. The string $\beta = 1000$ corresponds to a negative integer with a $\{0, \bar{1}\}$-radix 2 representation that has the prefix $0\bar{1}\bar{1}\bar{1}$. The string $0\bar{1}\bar{1}\bar{1}$ is calculated from β by subtracting a string of ones. The closest element in $\mathcal{C}_3$ to a negative integer with this prefix is

$$(\bar{1}000\ldots0)_2 = -1 \cdot 2^{\ell}.$$

Hence the prefix 1000 is mapped to the string $\bar{1}00$.

If we initialize s to 0 and slide a 4-digit window across the $\{0,1\}$-radix 2 representation of 379, we construct a new representation as follows:

$$\begin{array}{r|l}
379 = (\underline{\overset{0}{0}101}111011.000)_2 & 003 \\
(010\underline{\overset{1}{1}111}011.000)_2 & 0030 \\
(0101\underline{\overset{1}{1}110}11.000)_2 & 00300 \\
(01011\underline{\overset{1}{1}101}1.000)_2 & 003000 \\
(010111\underline{\overset{1}{1}011}.000)_2 & 00300001\bar{1}0 \\
(010111101\underline{\overset{1}{1}.000})_2 & 0030000\bar{1}0\bar{1}.000
\end{array}$$

As in our previous examples, each time the contents of the window and the value of s match an entry in the left hand column of our table we output the corresponding 3-digit string, update s and then advance the window 3 digits to the right. Otherwise, we output a single 0, leave s unchanged and advance the window 1 digit to the right.

Note that this implementation is well defined; that is, no matter what the input, there is never any ambiguity about what sequence of steps to take. This is true even when the input has more than one closest element of $\mathcal{C}_3$. The strings β' in table always select the closest element of $\mathcal{C}_3$ that has the largest absolute value. In fact, any implementation of Algorithm 2.1 based on examining finite length prefixes of the $\{0,1\}$-radix 2 representation must use this strategy for all but a finite number of inputs (e.g., when $w = 3$, consider inputs of the form $n = 5 \cdot 2^i$).

It is not difficult to construct a similar table for the case $w = 2$. Doing so results in an implementation of Algorithm 2.1 which coincides with the algorithm Joye and Yen give in [5].

4.3 A New Window Method

The look-up table of the previous section can be modified as follows:

β	s	β'	s'
0100	0	01	0
0101	0	003	1
0110	0	003	0
0111	0	1	1
1000	1	$\bar{1}$	0
1001	1	$00\bar{3}$	1
1010	1	$00\bar{3}$	0
1011	1	$0\bar{1}$	1

In this table, the strings β' are all of the form $0^j d$ where d is a nonzero digit in D_3. Because the strings β' now have different lengths we must account for this as we slide the window across the $\{0,1\}$-radix 2 representation of our input. The table can also be represented as follows:

β	s	j	d	s'
0100	0	1	1	0
0101	0	2	3	1
0110	0	2	3	0
0111	0	0	1	1
1000	1	0	$\bar{1}$	0
1001	1	2	$\bar{3}$	1
1010	1	2	$\bar{3}$	0
1011	1	1	$\bar{1}$	1

This version of the table is better suited for use with Algorithm 1.1.

Combining our implementation of Algorithm 2.1 with Algorithm 1.1 results in the following window method for scalar multiplication:

```
Algorithm 4.3: $w$-MSF-WINDOW-METHOD$(n, P)$

comment: $w \geq 2,\ D_w = \{0\} \cup \{d \in \mathbb{Z} : d \text{ odd}, |d| < 2^{w-1}\}$
         $n = (b_{\ell-1} \dots b_1 b_0)_2,$ where $b_i \in \{0,1\}$
         $\beta = b_{i+1} b_i \dots b_{i-w+1}$
external $T : (\beta, s) \mapsto (j, d, s)$

for each $d \in D_w$ with $d > 0$
  do $P_d \leftarrow dP$
$Q \leftarrow \infty,\quad i \leftarrow \ell - 1,\quad s \leftarrow 0$
while $i \geq 0$
  do $(j, d, s) \leftarrow T(\beta, s)$
     $Q \leftarrow 2^j Q$
     if $d \neq 0$
       then if $d > 0$
              then $Q \leftarrow Q + P_d$
              else $Q \leftarrow Q - P_{-d}$
     $i \leftarrow i - j$
return $Q$
```

5 Remarks

In proving that our new representations have a minimal number of nonzero digits, we essentially dealt with the following two statements concerning odd integers:

$$\mathsf{wt}^*(n) = 1 + \mathsf{wt}^*((n-r)/2) \quad \text{where } r = n \text{ mods } 2^w \tag{9}$$

$$\mathsf{wt}^*(n) = 1 + \mathsf{wt}^*(n-c) \quad \text{where } c \in \mathcal{C}_w \text{ is closest to } n. \tag{10}$$

In our proof, we noted that (9) is true (by the minimality of the w-NAF) and then showed that (9) implies (10). The same arguments can be used to show that (10) implies (9). Thus, (9) and (10) are logically equivalent, which is perhaps surprising.

One theme which seems to recur in this work is the idea of choosing a closest element. This a simple strategy but, as we have seen, it is also sometimes an optimal strategy.

The w-NAF has a very simple combinatorial description. From this description, it is very easy to look at a representation and quickly decide whether or not it is a w-NAF. For our new representations, this does not appear to be quite so easy.

References

1. I. F. Blake, G. Seroussi and N. P. Smart. *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.
2. H. Cohen, A. Miyaji and T. Ono. Efficient Elliptic Curve Exponentiation Using Mixed Coordinates, in "Advances in Cryptology – ASIACRYPT '98", *Lecture Notes in Computer Science* **1514** (1998), 51–65.
3. D. M. Gordon. A Survey of Fast Exponentiation Methods, *Journal of Algorithms* **27** (1998), 129–146.
4. D. Hankerson, A. Menezes and S. Vanstone. *Guide to Elliptic Curve Cryptography*, Springer, 2004.
5. M. Joye and S. Yen. Optimal Left-to-Right Binary Signed-Digit Recoding, *IEEE Transactions on Computers* **49** (2000), 740–748.
6. D. E. Knuth, *Seminumerical Algorithms*, Volume 2 of *The Art of Computer Programming*, Addison-Wesley, Third edition, 1997.
7. J. A. Muir and D. R. Stinson. Minimality and Other Properties of the Width-w Nonadjacent Form. Technical Report CORR 2004-08, Centre for Applied Cryptographic Research. Available from `http://www.cacr.math.uwaterloo.ca/-techreports/2004/corr2001-08.ps`.
8. V. Müller. Fast Multiplication on Elliptic Curves over Small Fields of Characteristic Two. *Journal of Cryptology* **11** (1998), 219–234.
9. J. A. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography* **19** (2000), 195–249.