Fast Elliptic Curve Cryptography Using Optimal Double-Base Chains

Vorapong Suppakitpaisarn[1,2], Masato Edahiro[1,3], and Hiroshi Imai[1,2]

[1]Graduate School of Information Science and Technology, the University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
[2]ERATO-SORST Quantum Computation and Information Project, JST
5-28-3 Hongo, Bunkyo-ku, Tokyo, Japan
[3]System IP Core Research Laboratories, NEC Corporation
1753 Shimonumabe, Nakahara-ku, Kawasaki, Japan

Abstract. In this work, we propose an algorithm to produce the double-base chains that optimize the time used for computing an elliptic curve cryptosystem. The double-base chains is the representation that combining the binary and ternary representation. By this method, we can reduce the Hamming weight of the expansion, and reduce the time for computing the scalar point multiplication ($Q = rS$), that is the bottleneck operation of the elliptic curve cryptosystem. This representation is very redundant, i.e. we can present a number by many expansions. Then, we can select the way that makes the operation fastest. However, the previous works on double-bases chain have used a greedy algorithm, and their solutions are not optimized. We propose the algorithm based on the dynamic programming scheme that outputs the optimized the double-bases chain. The experiments show that we have reduced the time for computing the scalar multiplication by 3.88-3.95%, the multi-scalar multiplication by 2.55-4.37%, and the multi-scalar multiplication on the larger digit set by 3.5-12%.

Key words: Elliptic Curve Cryptography, Minimal Weight Conversion, Digit Set Expansion, Double-Base Chains

1 Introduction

Scalar multiplication is the bottleneck operation of the elliptic curve cryptography. It is to compute

$$Q = rS$$

when S, Q are points on the elliptic curve and r is a positive integer. There are many works proposed the ways to reduce the computation time of the operation. Most of them are based on double-and-add method. This method depends on the binary expansion of r explained as follows:

Define $n = \lfloor \lg r \rfloor$, and

$$r = \sum_{t=0}^{n-1} r_t 2^t$$

where r_t is a member of a finite set Ds. We call Ds as *digit set*, and $R = \langle r_0, r_1, \dots, r_{n-1} \rangle$ as the *binary expansion* of r. The Hamming weight $W(R)$ is defined as

$$W(R) = \sum_{t=0}^{n-1} W(r_t),$$

where $W(r_t) = 0$ when $r_t = 0$ and $W(r_t) = 1$ otherwise. For example, let $Ds = \{0, 1\}$, and $r = 127 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6$. The binary expansion of r is $R = \langle 1, 1, 1, 1, 1, 1, 1 \rangle$, and the Hamming weight $W(R) = 7$.

In the double-and-add scheme, we need two elementary operations, that are point doubles($S + S$, $2S$) and point additions($S + Q$ when $S \neq Q$). The number of point doubles is constant for each scalar r. However, the number of point additions depends on the binary expansion. In some Ds, there are more than one way to expand a positive integer, and we need select the efficient way. This problem has been studied extensively in [1, 2].

In [3, 4], Dimitrov et al. proposed to use double-base chains on the elliptic curve cryptography. Let

$$r = \sum_{t=0}^{m-1} r_t 2^{x_t} 3^{y_t},$$

such that r_t be a member of digit set $Ds - \{0\}$ and $x_t \leq x_{t+1}$, $y_t \leq y_{t+1}$ for all t. We define

$$C[r] = \langle R, X, Y \rangle,$$

when $R = \langle r_0, r_1, \dots, r_{m-1} \rangle$, $X = \langle x_0, x_1, \dots, x_{m-1} \rangle$, $Y = \langle y_0, y_1, \dots, y_{m-1} \rangle$ as the double-base chains of r. Also, we define the Hamming weight of double-base chains $W(C[r]) = m$. For examples, one of the double-base chains of $127 = 2^0 3^0 + 2^1 3^2 + 2^2 3^3$ is $C[127] = \langle R, X, Y \rangle$ when

$$R = \langle 1, 1, 1 \rangle,$$

$$X = \langle 0, 1, 2 \rangle,$$

$$Y = \langle 0, 2, 3 \rangle.$$

In this case $W(C[127]) = 3$.

In addition to point doubles and point additions needed in the binary expansion, we also need point triples($3S$). In some elliptic curves where the point triple is relatively fast, double-base chains are shown to be faster than the binary expansion.

Similar to the binary expansion, every scalars have more than one double-base chains, and the efficiency of elliptic curve strongly depends on which chain we use. The algorithm to select the good double-base chains is very important. There are many works have studied the problem [3–6], and proposed greedy algorithms that cannot guarantee the best chain. On the other hand, we adapted our previous works [7–9], where we propose the dynamic programming algorithm to find the minimal weight expansion of various representation. Then, we can find the algorithm that always outputs the best chain, where the computation time of all elementary operations (point additions, point doubles, point triples) have been considered. By the experiment, we have shown that the optimal double-base chains are better than the best greedy algorithm proposed on double base chain [6] by 3.9% when $Ds = \{0, \pm 1\}$.

Recently, there is the independent work [10] proposed the algorithm which can output the chains with least Hamming weight when $Ds = \{0, 1\}$. We consider their work as the specific case of our works as we are working on any finite digit sets. Also, our algorithm can output the least Hamming weight by adjusting the computation time for point doubles and point additions to zero. When the point addition is the only elementary operation concerned, minimizing the computation time of the scalar multiplication means optimizing the Hamming weight.

There is also the work utilizing Yao's algorithm with *double base number system* [11, 12], which is the double-base without the restriction such that $x_t \leq x_{t+1}$ and $y_t \leq y_{t+1}$ [13]. Their results of the algorithm is comparable to our results even when we select the Ds that gives the best result. However, our algorithm works better on the elliptic curve that the point triple is fast comparing to the point double. These include inverted coordinates on edwards curves which has the fastest point doubles [14] up to this states.

Our contribution also covers the multi-scalar multiplication, that is the operation used in digital signature scheme,

$$Q = r_1 S_1 + r_2 S_2 + \cdots + r_d S_d.$$

To work on the operation, we need to define the joint binary expansion, and the joint Hamming weight. Let $n = \lfloor \lg(\max(r_1, \ldots, r_d)) \rfloor$. We define the joint binary expansion of $\langle r_1, \ldots, r_d \rangle$ as $\langle R_1, \ldots, R_d \rangle$ where $R_i = \langle r_{i,0}, \ldots, r_{i,n-1} \rangle$ is the binary expansion of r_i. Let

$$w_t = \begin{cases} 0 & \text{if } \langle r_{1,t}, \ldots, r_{d,t} \rangle = \langle \mathbf{0} \rangle, \\ 1 & \text{otherwise.} \end{cases}$$

We can define the joint Hamming weight $JW(R_1, \ldots, R_d)$ as

$$JW(R_1, \ldots, R_d) = \sum_{t=0}^{n-1} w_t.$$

For example, one of the joint binary expansion of $r_1 = 127 = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6$ and $r_2 = 109 = 2^0 + 2^2 + 2^3 + 2^5 + 2^6$ is

$$R_1 = \langle 1, 1, 1, 1, 1, 1, 1 \rangle,$$

$$R_2 = \langle 1, 0, 1, 1, 0, 1, 1 \rangle.$$

The joint Hamming weight $JW(R_1, R_2)$ is 7.

For the elliptic curve cryptography, the optimal joint binary expansion is the expansion with the least joint Hamming weight. This problem has been studied extensively in [7–9, 15].

For the double-base chains, we define the joint double-base chains

$$C[r_1, \ldots, r_d] = \langle R_1, \ldots, R_d, X, Y \rangle$$

where

$$R_i = \langle r_{i,0}, r_{i,1}, \ldots, r_{i,m-1} \rangle,$$

$$X = \langle x_0, x_1, \ldots, x_{m-1} \rangle,$$

$$Y = \langle y_0, y_1, \ldots, y_{m-1} \rangle,$$

$$r_i = \sum_{t=0}^{m-1} r_{i,t} 2^{x_t} 3^{y_t},$$

$\langle r_{1,t}, \ldots, r_{d,t} \rangle \in Ds - \{\langle \mathbf{0} \rangle\}$, $x_t \leq x_{t+1}$, and $y_t \leq y_{t+1}$. Similar to the Hamming weight for the double-base chains, the joint Hamming weight of the joint double-base chains is $JW(C) = m$. For example, one of the joint double-base chain of $r_1 = 127 = 2^0 3^0 + 2^1 3^2 + 2^2 3^3$ and $r_2 = 109 = 2^0 3^0 + 2^2 3^3$ is $C[127, 109] = \langle R_1, R_2, X, Y \rangle$ where

$$R_1 = \langle 1, 1, 1 \rangle,$$

$$R_2 = \langle 1, 0, 1 \rangle,$$

$$X = \langle 0, 1, 2 \rangle,$$

$$Y = \langle 0, 1, 3 \rangle.$$

The joint Hamming weight $JW(C)$ is 3.

The optimal binary expansion on this case has been studied in [7–9], and the greedy algorithm for double-base chains has been proposed in [14, 16, 17]. We generalize the idea on scalar multiplication and propose the optimal double-base chain for multi-scalar multiplication. Our algorithm improve the greedy algorithm proposed in [14] for $Ds = \{0, \pm 1\}$ and $d = 2$ by 2.81% on 192-bit scalar and 4.37% on 512-bit scalar. We also reduce the computation time of the representation such that $Ds = \{0, \pm 1, \pm 5\}$ proposed in the same paper by 4.1% on 192-bit scalar and 5.1% on 512-bit scalar. Moreover, we compare our results with the hybrid binary-ternary representation proposed in [16, 17] which are using $Ds = \{0, \pm 1, \pm 2, 3\}$ and $d = 2$. On the same representation, our results are better by 12%.

This paper is organized as follows: we show the double-and-add scheme, and how we utilize the double-base chain to elliptic curve cryptography in Section 2. In Section 3, we show our algorithm which outputs the optimal double-base chain. Next, we show the experimental results comparing to the existing works in Section 4. Last, we conclude the paper in Section 5.

2 Preliminaries

Using the binary expansion $R = \langle r_0, r_1, \ldots, r_{n-1}\rangle$, where $r = \sum_{t=0}^{n-1} r_t 2^t$ explained in Section 1, we can compute the scalar multiplication $Q = rS$ by double-and-add scheme as shown in Algorithm 1. For example, we compute $Q = 127S$ when the binary expansion of 127 is $R = \langle 1, 1, 1, 1, 1, 1, 1\rangle$ as follows:

$$Q = 2(2(2(2(2(2S + S) + S) + S) + S) + S) + S.$$

We need six point doubles and six point additions in this example. Generally, we need $n - 1$ point doubles, and n point additions. However, Q is initialized to O, and we need not the point addition on the first iteration. Also, $r_t S = 0$ if $r_t = 0$, and we need not the point addition in this case. Hence, the number of the point additions is $W(R) - 1$, where $W(R)$ the Hamming weight of the expansion defined in Section 1. The Hamming weight tends to be less if the digit set Ds is larger. However, as we need to precompute $r_t S$ for all $r_t \in Ds$, using big Ds makes cost for the precomputation higher.

Algorithm 1 Double-and-add method

Require: A point on elliptic curve S, the positive integer r with the binary expansion $\langle r_0, r_1, \ldots, r_{n-1}\rangle$.
Ensure: $Q = rS$

```
Q ← O
for t ← n − 1 downto 0 do
    Q ← Q + r_t S
    if t ≠ 0 then
        Q ← 2Q
    end if
end for
```

In Algorithm 2, we show how to apply the double-base chain

$$C[r] = \langle R, X, Y\rangle,$$

when

$$R = \langle r_0, r_1, \ldots, r_{m-1}\rangle,$$

$$X = \langle x_0, x_1, \ldots, x_{m-1}\rangle,$$

$$Y = \langle y_0, y_1, \ldots, y_{m-1}\rangle$$

to compute scalar multiplication. For example, one of the double-base chain of $127 = 2^0 3^0 + 2^1 3^2 + 2^2 3^3$ is $C[127] = \langle R, X, Y\rangle$, where $R = \langle 1, 1, 1\rangle$, $X = \langle 0, 1, 2\rangle$, $Y = \langle 0, 1, 3\rangle$. Hence, we can compute $Q = 127S$ as follows:

$$Q = 2^1 3^2 (2^1 3^1 S + S) + S.$$

In this case, we need two point additions, two point doubles, and three point triples. In general, the number of point additions is $W(C) - 1 = m - 1$ defined in Section 1. On the other hand, the number of point doubles and point triples are x_{m-1} and y_{m-1} respectively.

Algorithm 2 Using the double-base chain to compute scalar multiplication

Require: A point on elliptic curve S, the positive integer r with the double-base chains $C[r] = \langle R, X, Y \rangle$, where $R = \langle r_0, \ldots, r_{m-1} \rangle$, $X = \langle x_0, \ldots, x_{m-1} \rangle$, $Y = \langle y_0, \ldots, y_{m-1} \rangle$.
Ensure: $Q = rS$

$Q \leftarrow O$
for $t \leftarrow m - 1$ downto 0 **do**
  $Q \leftarrow Q + r_t S$
  if $t \neq 0$ **then**
    $Q \leftarrow 2^{(x_{t-1} - x_t)} 3^{(y_{t-1} - y_t)} Q$
  else
    $Q \leftarrow 2^{x_0} 3^{y_0} Q$
  end if
end for

In the double-and-add method, the number of point doubles required is proved to be constantly equal to $n - 1 = \lfloor \lg r \rfloor - 1$. Then, the efficiency of the binary expansion strongly depends on the number of point additions or the Hamming weight. However, the number of point doubles and point triples are not constant, as discussed in the previous paragraph that they are equal to x_{m-1} and y_{m-1} respectively. Hence, we need to optimize the value

$$x_{m-1} \cdot P_{dou} + y_{m-1} \cdot P_{tri} + (W(C[r]) - 1) \cdot P_{add},$$

when $P_{dou}, P_{tri}, P_{add}$ are the cost for point double, point triple, and point addition respectively. This is different from the literature [1, 2, 7–10] where only the Hamming weight is considered.

To compute multi-scalar multiplication $Q = r_1 S_1 + \cdots + r_d S_d$, we can utilize Shamir's trick [18]. Using the trick, the operation is claimed to be faster than to compute $r_1 S_1, \ldots, r_d S_d$ separately and add them together. We show the Shamir's Trick in Algorithm 3. For example, we can compute $Q = r_1 S_1 + r_2 S_2 = 127 S_1 + 109 S_2$ given the expansion of r_1, r_2 are $R_1 = \langle 1, 1, 1, 1, 1, 1, 1 \rangle$ and $R_2 = \langle 1, 0, 1, 1, 0, 1, 1 \rangle$ as follows:

$$127 S_1 + 109 S_2 = 2(2(2(2(2(2D + D) + S_1) + D) + D) + S_1) + D,$$

where $D = S_1 + S_2$. We need six point doubles, six point additions in this case. Generally, we need $\lfloor \lg(\max(r_1, \ldots, r_d)) \rfloor - 1$ point doubles, and $JW(R_1, R_2)$ point additions.

Algorithm 4 shows how we apply Shamir's trick with the joint double-base chain defined in Section 1. $Q = 127 S_1 + 109 S_2 = (2^0 3^0 + 2^1 3^2 + 2^2 3^3) S_1 + (2^0 3^0 +$

Algorithm 3 Shamir's trick with the joint binary expansion

Require: A point on elliptic curve S, the positive integer $r_1, \ldots, r_d$ with the binary expansion $R_1 = \langle r_{1,0}, r_{1,1}, \ldots, r_{1,n-1} \rangle, \ldots, R_d = \langle r_{d,0}, r_{d,1}, \ldots, r_{d,n-1} \rangle$.
Ensure: $Q = \sum_{i=1}^{d} r_i S_i$

```
Q ← O
for t ← n − 1 downto 0 do
    Q ← Q + Σ_{i=1}^{d} r_{i,t} S_i
    if t ≠ 0 then
        Q ← 2Q
    end if
end for
```

$2^2 3^3)S_2$ can be computed as follows:

$$Q = 127S_1 + 109S_2 = 2^1 3^2 (2^1 3^1 D + S_1) + D,$$

where $D = S_1 + S_2$. Hence, we need two point additions, two point doubles, and three point triples. Similar to the double-base chain, x_{m-1} point doubles, y_{m-1} point triples, and $JW(C[127, 109]) - 1$ point additions are required in general.

Algorithm 4 Using Shamir's trick with the joint double-base chain

Require: A point on elliptic curve S, the positive integer r with the double-base chains $C[r_1, \ldots, r_d] = \langle R_1, \ldots, R_d, X, Y \rangle$, where $R_i = \langle r_{i,0}, \ldots, r_{i,m-1} \rangle$, $X = \langle x_0, \ldots, x_{m-1} \rangle$, $Y = \langle y_0, \ldots, y_{m-1} \rangle$.
Ensure: $Q = rS$

```
Q ← O
for t ← m − 1 downto 0 do
    Q ← Q + Σ_{i=1}^{d} r_{i,t} S_i
    if t ≠ 0 then
        Q ← 2^(x_{t−1} − x_t) 3^(y_{t−1} − y_t) Q
    else
        Q ← 2^(x_0) 3^(y_0) Q
    end if
end for
```

3 Algorithm

3.1 Algorithm for single integer with $Ds = \{0, 1\}$

Define the cost to compute r using the chain $C[r] = \langle R, X, Y \rangle$ as

$$P(C[r]) = \begin{cases} 0 & \text{if } C[r] = \langle \langle\rangle, \langle\rangle, \langle\rangle \rangle, \\ x_{m-1} \cdot P_{dou} + y_{m-1} \cdot P_{tri} + (W(C[r]) - 1) \cdot P_{add} & \text{otherwise.} \end{cases}$$

Our algorithm is to find the double-base chain of r, $C[r] = \langle R, X, Y \rangle$ such that for all double-base chain of r, $Ce[r] = \langle Re, Xe, Ye \rangle$,

$$P(Ce[r]) \geq P(C[r]).$$

To explain the algorithm, we start with a small example explained in Example 1 and Figure 1.

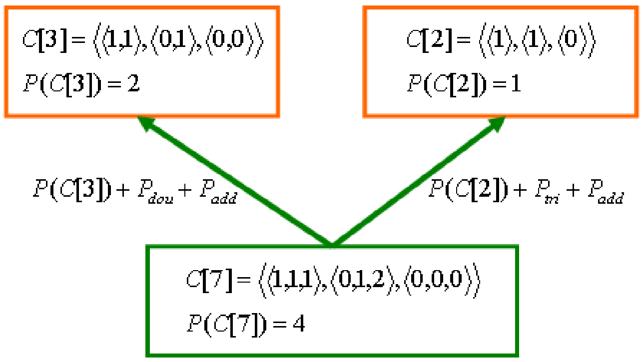

Fig. 1. We can compute $C[7]$ by two ways. The first way is to compute $C[3]$, and perform a point double and a point addition. The cost in this way is $P(C[3]) + P_{dou} + P_{add}$. The second way is to compute $C[2]$, and perform a point triple and a point addition, where the cost is $P(C[2] + P_{tri} + P_{add})$. The cost of the first way is smaller than the second way, and we select the first way to compute $C[7]$.

Example 1. Find the optimal chain $C[7] = \langle R, X, Y \rangle$ given $Ds = \{0, 1\}$, $P_{tri} = 20$, $P_{dou} = 1$, and $P_{add} = 1$.

Assume that we are given the optimal chain $C[3] = \langle R[3], X[3], Y[3] \rangle$ of $3 = \lfloor \frac{7}{2} \rfloor$ and $C[2] = \langle R[2], X[2], Y[2] \rangle$ of $2 = \lfloor \frac{7}{3} \rfloor$. We want to rewrite 7 as

$$7 = \sum_{t=0}^{m-1} r_t 2^{x_t} 3^{y_t},$$

when $r_t \in Ds - \{0\} = \{1\}$. As $2 \nmid 7$ and $3 \nmid 7$, the smallest term much be $1 = 2^0 3^0$. Hence, $x_0 = 0$ and $y_0 = 0$. Then,

$$7 = \sum_{t=1}^{m-1} 2^{x_t} 3^{y_t} + 1.$$

By this equation, there are only two ways to compute the scalar multiplication $Q = 7S$ with Algorithm 2. The first way is to compute $3S$, do point double to $6S$ and point addition to $7S$. As we know the the optimal chain for 3, the cost using this way is

$$P(C[3]) + P_{dou} + P_{add}.$$

The other way is to compute $2S$, do point triple to $6S$ and point addition to $7S$. In this case, the cost is

$$P(C[2]) + P_{tri} + P_{add}.$$

The optimal way is to select one of these two ways. We will show later that $P(C[3]) = 2$ and $P(C[2]) = 1$. Then,

$$P(C[3]) + P_{dou} + P_{add} = 2 + 1 + 1 = 4,$$

$$P(C[2]) + P_{tri} + P_{add} = 1 + 20 + 1 = 22.$$

We select the first choice, and the optimal cost is $P(C[7]) = 4$. The optimal $C[7] = \langle R, X, Y\rangle$ is considered as follows:

$$R = \langle 1, R[3]\rangle.$$

$$X = \langle x_0, \ldots, x_{m-1}\rangle,$$

where $x_0 = 0$ and $x_t = x[3]_{t-1} + 1$ for $1 \le t \le m-1$.

$$Y = \langle y_0, \ldots, y_{m-1}\rangle,$$

where $y_0 = 0$ and $y_t = y[3]_{t-1}$ for $1 \le t \le m-1$.

Next, we find $C[3]$ that is the optimal double-base chain of $\left\lfloor \frac{7}{2} \right\rfloor = 3$. Similar to $7S$, we can compute $3S$ by two ways. The first way is to triple the point S. Using this way, we need one point triple, which costs $P_{tri} = 20$. The double-base chain in this case will be

$$\langle\langle 1\rangle, \langle 0\rangle, \langle 1\rangle\rangle.$$

The other way is that we double point S to $2S$, then add $2S$ with S to get $3S$. The cost is $P_{dou} + P_{add} = 1 + 1 = 2$. In this case, the double-base chain is

$$\langle\langle 1, 1\rangle, \langle 0, 1\rangle, \langle 0, 0\rangle\rangle.$$

We select the better double-base chain that is

$$C[3] = \langle\langle 1, 1\rangle, \langle 0, 1\rangle, \langle 0, 0\rangle\rangle.$$

Last, we find $C[2]$, the optimal double-base chain of $\left\lfloor \frac{7}{3} \right\rfloor = 2$. The interesting point to note is that there are only one choice to consider in this case. This is because the fact that we cannot rewrite 2 by $3A + B$ when $A \in \mathbb{Z}$ and $B \in Ds$ if $r \equiv 2 \bmod 3$. Then, the only choice left is to double the point S, which costs 1, and the double-base chain is

$$C[2] = \langle\langle 1\rangle, \langle 1\rangle, \langle 0\rangle\rangle.$$

To conclude, the optimal double-base chain for 7 in this case is

$$C[7] = \langle\langle 1,1,1\rangle, \langle 0,1,2\rangle, \langle 0,0,0\rangle\rangle.$$

We note that C is not the double-base with the least Hamming weight as

$$Ce[7] = \langle\langle 1,1\rangle, \langle 0,1\rangle, \langle 0,1\rangle\rangle$$

has lower Hamming weight.

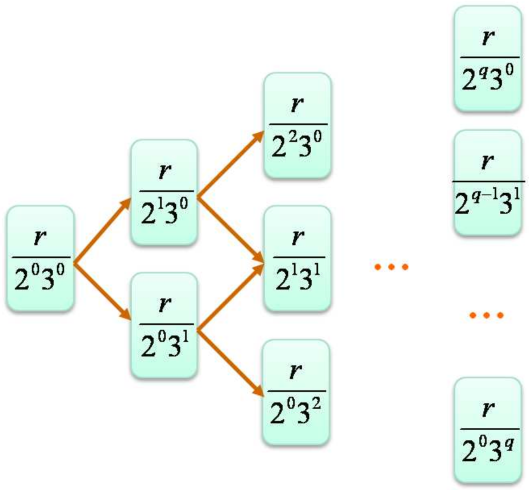

Fig. 2. Bottom-up algorithm to find the optimal double-base chain of r

Define $C[r], C[\lfloor \frac{r}{2} \rfloor], C[\lfloor \frac{r}{3} \rfloor]$ be the optimal double-base chain of r, $\lfloor \frac{r}{2} \rfloor$, $\lfloor \frac{r}{3} \rfloor$ respectively.From Example 1, we consider r as six cases as follows:

1. $r \equiv 0 \bmod 2$ and $r \equiv 0 \bmod 3$ ($r \equiv 0 \bmod 6$). In this case, we have two choices. The first choice is to use the chain $C[\lfloor \frac{r}{2} \rfloor]$ and do the point double. The other choice is to use the chain $C[\lfloor \frac{r}{3} \rfloor]$ and do the point triple.
2. $r \equiv 1 \bmod 2$ and $r \equiv 1 \bmod 3$ ($r \equiv 1 \bmod 6$). In this case, we have two choices. The first choice is to use the chain $C[\lfloor \frac{r}{2} \rfloor]$ and do the point double with point addition. The other choice is to use the chain $C[\lfloor \frac{r}{3} \rfloor]$ and do the point triple with point addtion.
3. $r \equiv 0 \bmod 2$ and $r \equiv 2 \bmod 3$ ($r \equiv 2 \bmod 6$). In this case, we have only one choice. That is do the point double on $C[\lfloor \frac{r}{2} \rfloor]$.

4. $r \equiv 1 \bmod 2$ and $r \equiv 0 \bmod 3$ ($r \equiv 3 \bmod 6$). In this case, we have two choices. The first choice is to use the chain $C[\lfloor \frac{r}{2} \rfloor]$ and do the point double with point addition. The other choice is to use the chain $C[\lfloor \frac{r}{3} \rfloor]$ and do the point triple.
5. $r \equiv 0 \bmod 2$ and $r \equiv 1 \bmod 3$ ($r \equiv 4 \bmod 6$). In this case, we have two choices. The first choice is to use the chain $C[\lfloor \frac{r}{2} \rfloor]$ and do the point double. The other choice is to use the chain $C[\lfloor \frac{r}{3} \rfloor]$ and do the point triple with point addition.
6. $r \equiv 1 \bmod 2$ and $r \equiv 2 \bmod 3$ ($r \equiv 5 \bmod 6$). In this case, we have only one choice. That is do the point double on $C[\lfloor \frac{r}{2} \rfloor]$. Then, do the point addition.

We summarize the above statement as the following relation:

$$P(C^r) = \begin{cases} \min(P(C[\lfloor \frac{r}{2} \rfloor]) + P_{dou}, P(C[\lfloor \frac{r}{3} \rfloor]) + P_{tri}) & r \equiv 0 \bmod 6 \\ \min(P(C[\lfloor \frac{r}{2} \rfloor]) + P_{dou} + P_{add}, P(C[\lfloor \frac{r}{3} \rfloor]) + P_{tri} + P_{add}) & r \equiv 1 \bmod 6 \\ P(C[\lfloor \frac{r}{2} \rfloor]) + P_{dou} & r \equiv 2 \bmod 6 \\ \min(P(C[\lfloor \frac{r}{2} \rfloor]) + P_{dou} + P_{add}, P(C[\lfloor \frac{r}{3} \rfloor]) + P_{tri}) & r \equiv 3 \bmod 6 \\ \min(P(C[\lfloor \frac{r}{2} \rfloor]) + P_{dou}, P(C[\lfloor \frac{r}{3} \rfloor]) + P_{tri} + P_{add}) & r \equiv 4 \bmod 6 \\ P(C[\lfloor \frac{r}{2} \rfloor]) + P_{dou} + P_{add} & r \equiv 5 \bmod 6 \end{cases}$$

Next, we will step to the algorithm. In Example 1, we consider the computation as a top-down algorithm. However, bottom-up algorithm is better way to implement the idea. We begin the algorithm by computing the double-base chain of $\lfloor \frac{r}{2^x 3^y} \rfloor$ for all $x, y \in \mathbb{Z}^+$ such that $x + y = q$ where $2^q \leq r < 2^{q+1}$. Then, we move to compute the double-base chain of $\lfloor \frac{r}{2^x 3^y} \rfloor$ for all $x, y \in \mathbb{Z}^+$ such that $x + y = q - 1$ by referring to the double-base chain of $\lfloor \frac{r}{2^x 3^y} \rfloor$ when $x + y = q$. We decrease the number $x + y$ until $x + y = 0$, and we get the chain of $r = \lfloor \frac{r}{2^0 3^0} \rfloor$. We illustrate this idea in Figure 2.

We use this idea with the equation above to propose Algorithm 5,6. In the algorithm, we denote $C[r] = \langle R[r], X[r], Y[r] \rangle$ as the optimal double-base chain of r. The algorithm completes in $O(\lfloor \lg^2 r \rfloor)$, and consumes $O(\lfloor \lg^2 r \rfloor)$ memory. ($O(\lfloor \lg r \rfloor)$ chains, each chain needs $O(\lfloor \lg r \rfloor)$) The algorithm is obviously following the idea given above. We explain how the algorithm works by Example 2.

Example 2. Find the optimal expansion of 10 when $P_{add} = 1$, $P_{dou} = 1$, and $P_{tri} = 1$.

In this case, the value q assigned in Line 1 is $\lfloor \lg 10 \rfloor = 3$.

- On this first loop when $q \leftarrow 3$, the possible (x, y) are $(3, 0), (2, 1), (1, 2), (0, 3)$, and the value $v \leftarrow \lfloor \frac{r}{2^x 3^y} \rfloor$ are $\lfloor \frac{10}{2^3 3^0} \rfloor = 1$, $\lfloor \frac{10}{2^2 3^1} \rfloor = 0$, $\lfloor \frac{10}{2^1 3^2} \rfloor = 0$, $\lfloor \frac{10}{2^0 3^3} \rfloor = 0$. By Lines 5-8 of Algorithm 5, the optimal expansion of 0 is $\langle\langle\rangle, \langle\rangle, \langle\rangle\rangle$, and the optimal expansion of 1 is $\langle\langle 1 \rangle, \langle 0 \rangle, \langle 0 \rangle\rangle$.
- We move to the second step when $q \leftarrow 2$ when the possible (x, y) are $(2, 0), (1, 1), (0, 2)$. In this case, the value v are $\lfloor \frac{10}{2^2 3^0} \rfloor = 2$, $\lfloor \frac{10}{2^1 3^1} \rfloor = 1$,

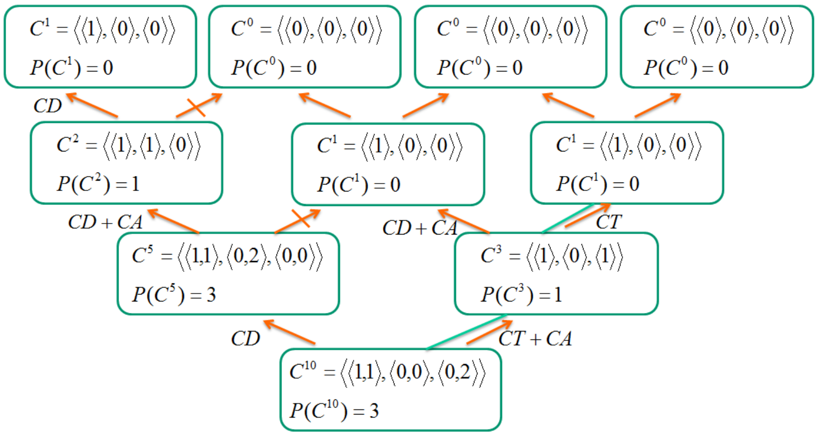

Fig. 3. Illustration of the proposed algorithm when the input $r = 10$, $P_{dou} = P_{tri} = P_{add} = 1$.

$\left\lfloor \frac{10}{2^0 3^2} \right\rfloor = 1$. As the first loop, $C[1] = \langle\langle 1\rangle, \langle 0\rangle, \langle 0\rangle\rangle$. We compute $C[2]$ using the *FindOptimal* method defined in Algorithm 6. In the algorithm, we compute c_2 in Lines 1-4. As

$$v_2 = \left\lfloor \frac{2}{1} \right\rfloor = 1,$$

$$\begin{aligned} P(C[v_2]) &= P(C[1]) = P(\langle\langle 1\rangle, \langle 0\rangle, \langle 0\rangle\rangle) \\ &= x_{m-1} \cdot P_{dou} + y_{m-1} \cdot P_{tri} + (W(C[1]) - 1) \cdot P_{add} \\ &= 0 \cdot P_{dou} + 0 \cdot P_{tri} + (1 - 1) \cdot P_{add} = 0. \end{aligned}$$

Then, $c_2 \leftarrow 0 + P_{dou} = 1$. We compute c_3 in Lines 5-10. In this case, $v \equiv 2 \bmod 3$, and $c_3 \leftarrow \infty$. Next, we compute $C[v]$ in Lines 11-19. Since $c_2 \leq c_3$ and $v \equiv 0 \bmod 2$, We edit $C[v_2] = \langle\langle 1\rangle, \langle 0\rangle, \langle 0\rangle\rangle$ to $C[v] = \langle\langle 1\rangle, \langle 1\rangle, \langle 0\rangle\rangle$ by Line 12.

– The third step is when $q \leftarrow 1$. The possible (x, y) are $(1, 0)$ and $(0, 1)$, and the value v are $\left\lfloor \frac{10}{2^1 3^0} \right\rfloor = 5$ and $\left\lfloor \frac{10}{2^0 3^1} \right\rfloor = 3$. We compute $C[5]$ and $C[3]$ using Algorithm 6. When $v = 5$, $v_2 = 2$ and $v_3 = 1$.

$$\begin{aligned} P(C[v_2]) &= P(C[2]) = P(\langle\langle 1\rangle, \langle 1\rangle, \langle 0\rangle\rangle) \\ &= 1 \cdot P_{dou} + 0 \cdot P_{tri} + (1 - 1) \cdot P_{add} = 1. \end{aligned}$$

Then,

$$c_2 = P(C[v_2]) + P_{dou} + P_{add} = 1 + 1 + 1 = 3.$$

On the other hand, $c_3 = \infty$ as $5 \equiv 2 \bmod 3$. We edit

$$C[2] = \langle\langle 1\rangle, \langle 1\rangle, \langle 0\rangle\rangle$$

Algorithm 5 The algorithm finding the optimal double-base chain for single integer and $Ds = \{0, 1\}$

Require: the positive integer r
Ensure: the optimal double-base chain of r, $C[r] = \langle R[r], X[r], Y[r] \rangle$
1: $q \leftarrow \lfloor \lg r \rfloor$
2: **while** $q \geq 0$ **do**
3:     **for all** $x, y \in \mathbb{Z}^+$ such that $x + y = q$ **do**
4:         $v \leftarrow \left\lfloor \frac{r}{2^x 3^y} \right\rfloor$
5:         **if** $v = 0$ **then**
6:             $C[v] \leftarrow \langle \langle \rangle, \langle \rangle, \langle \rangle \rangle$
7:         **else if** $v = 1$ **then**
8:             $C[v] \leftarrow \langle \langle 1 \rangle, \langle 0 \rangle, \langle 0 \rangle \rangle$
9:         **else**
10:             $v_2 \leftarrow \left\lfloor \frac{r}{2^{x+1} 3^y} \right\rfloor$
11:             $v_3 \leftarrow \left\lfloor \frac{r}{2^x 3^{y+1}} \right\rfloor$
12:             $C[v] \leftarrow FindOptimal(v, C[v_2], C[v_3])$
13:         **end if**
14:     **end for**
15:     $q \leftarrow q - 1$
16: **end while**

to

$$C[5] = \langle \langle 1, 1 \rangle, \langle 0, 2 \rangle, \langle 0, 0 \rangle \rangle$$

by Line 14. When $v = 3$, $v_2 = v_3 = 1$, and

$$c_2 = P(C[1]) + P_{dou} + P_{add} = 0 + 1 + 1 = 2,$$

$$c_3 = P(C[1]) + P_{tri} = 1.$$

In this case, $c_3 < c_2$, and we edit

$$C[1] = \langle \langle 1 \rangle, \langle 0 \rangle, \langle 0 \rangle \rangle$$

as in Line 16. The result is

$$C[3] = \langle \langle 1 \rangle, \langle 0 \rangle, \langle 1 \rangle \rangle.$$

– In the last step, $q \leftarrow 0$, $(x, y) = (0, 0)$ and $v = 10$. Then, $v_2 = 5$ and $v_3 = 3$. As a result of the previous step,

$$C[v_2] = C[5] = \langle \langle 1, 1 \rangle, \langle 0, 2 \rangle, \langle 0, 0 \rangle \rangle,$$

and

$$P(C[5]) = 2 \cdot P_{dou} + 0 \cdot P_{tri} + (2 - 1) \cdot P_{add} = 3.$$

Then,

$$c_2 = 3 + P_{dou} = 4.$$

On the other hand,

$$C[v_3] = C[3] = \langle\langle 1\rangle, \langle 0\rangle, \langle 1\rangle\rangle.$$

$$P(C[3]) = 0 \cdot P_{dou} + 1 \cdot P_{tri} + (1-1) \cdot P_{add} = 1,$$

and

$$c_3 = 1 + P_{tri} + P_{add} = 3.$$

Hence, $c_3 < c_2$. As $10 \equiv 1 \bmod 3$, we edit $C[v_3] = C[3]$ as in Line 18. The result is

$$C[10] = \langle\langle 1, 1\rangle, \langle 0, 0\rangle, \langle 0, 2\rangle\rangle,$$

and the cost

$$P(C[10]) = 0 \cdot P_{dou} + 2 \cdot P_{tri} + (2-1) \cdot P_{add} = 3.$$

We illustrate this example as in Figure 3. In the figure, we represent the first step in the topmost row, and the last step in the bottommost. The lower node v refer to the result of the upper node v_2 and v_3 with some additional cost P_{dou}, P_{tri}, and P_{add}, and select the choice with minimal cost.

3.2 Generalized Algorithm for Any Digit Sets

In this section, we expand our results applying our former works [7–9] to our previous subsection. As a result, the proposed double-base chains can be used on the digit set other than $\{0, 1\}$.

When $Ds = \{0, 1\}$, we usually have two choices to compute $C[v]$. One is to perform a point double, and use the subsolution

$$C[v_2] = C[\left\lfloor \frac{v}{2} \right\rfloor].$$

Another is to perform a point triple, and use the subsolution

$$C[v_3] = C[\left\lfloor \frac{v}{3} \right\rfloor].$$

However, we have more choices when we deploy larger digit set. For example, when $Ds = \{0, \pm 1\}$

$$5 = 2 \times 2 + 1 = 3 \times 2 - 1 = 2 \times 3 - 1,$$

the number of cases increase from one in the previous subsection to three. Also, we need more optimal subsolution in this case. Even for point double, we need

$$C[2] = C[\left\lfloor \frac{5}{2} \right\rfloor]$$

and

$$C[3] = C[\left\lfloor \frac{5}{2} \right\rfloor + 1].$$

We call

$$v = \left\lfloor \frac{5}{2} \right\rfloor = 2$$

as *standard*, and the additional term $g[v]$ ($g[2] = 1$ in $C[3]$ and $g[2] = 0$ in $C[2]$) as *carry*. By this definition, we get the relation

$$(v \bmod 2) + g[v] = u + g[v_2],$$

when $u \in Ds$. This is the case when we choose to perform point double. Let

$$C[v_2 + g[v_2]] = \langle R', X', Y' \rangle$$

be the optimal solution of $r = v_2 + g[v_2]$. Define

$$X'' = \langle x''_0, \ldots, x''_{m-1} \rangle \text{ where } x''_i = x'_i + 1,$$
$$Y'' = Y',$$
$$R'' = R'.$$

The edited solution of $C[v + g[v]] = \langle R, X, Y \rangle$ when

$$X = \begin{cases} X'' & \text{if } u = 0, \\ \langle 0, X'' \rangle & \text{otherwise.} \end{cases}$$

$$Y = \begin{cases} Y'' & \text{if } u = 0, \\ \langle 0, Y'' \rangle & \text{otherwise.} \end{cases}$$

$$R = \begin{cases} R'' & \text{if } u = 0, \\ \langle u, R'' \rangle & \text{otherwise.} \end{cases}$$

If we perform point triple, the relation is

$$(v \bmod 3) + g[v] = u + g[v_3].$$

Again, we define $C[v_3 + g[v_3]] = \langle R', X', Y' \rangle$ be the optimal solution of $r = v_3 + g[v_3]$, and

$$X'' = X',$$
$$Y'' = \langle y''_0, \ldots, y''_{m-1} \rangle \text{ where } y''_i = y'_i + 1,$$
$$R'' = R'.$$

Similar to the case when we perform point double, the edited solution of $C[v + g[v]] = \langle R, X, Y \rangle$ when

$$X = \begin{cases} X'' & \text{if } u = 0, \\ \langle 0, X'' \rangle & \text{otherwise.} \end{cases}$$

$$Y = \begin{cases} Y'' & \text{if } u = 0, \\ \langle 0, Y'' \rangle & \text{otherwise.} \end{cases}$$

$$R = \begin{cases} R'' & \text{if } u = 0, \\ \langle u, R'' \rangle & \text{otherwise.} \end{cases}$$

The equation above is simply the generalized version of the equation defined in Algorithm 6 Lines 11-19, but in Algorithm 6 $g[v_2]$ is always equal to 0. In this case, $g[v_2]$ can be 1 as the example shown above. Then, we need to refer to the subsolution $C[v_2 + g[v_2]] = C[v_2 + 1]$, in addition to only $C[v_2]$ in Algorithm 6. Actually, $g[v_2]$ can be more than 0 and 1. We define the set of possible $g[v]$ for any v as G. Appendix A shows that G is always finite if Ds is finite, and proposes the algorithm to find the set. As we need refer to $C[v_2 + g[v_2]]$ for all $g[v_2] \in G$, we compute all $C[v + g[v]]$ for all $g[v] \in G$ to be referred when we compute the larger v. We illustrate the idea in Example 3 and Figure 4.

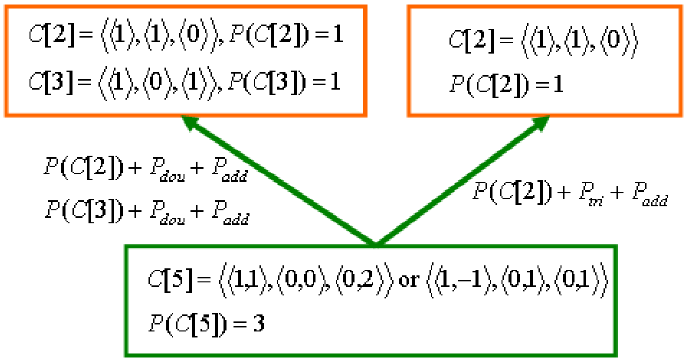

Fig. 4. Given $Ds = \{0, \pm 1\}$, we can compute $C[5]$ by three ways. The first way is to compute $C[2]$, and perform a point double and a point addition. The second is to compute $C[3]$, perform a point double, and a point substitution (add the point with $-S$). The third is to compute $C[2]$, perform a point triple, and a point substitution. All methods consume the same cost.

Example 3. Compute the optimal double-base chain of 5 when $P_{add} = P_{dou} = P_{tri} = 1$ and $Ds = \{0, \pm 1\}$.

When $Ds = \{0, \pm 1\}$, we can compute the carry set $G = \{0, 1\}$.

We want to compute $C[5] = \langle R, X, Y \rangle$ such that $r_i \in Ds$ and $x_i, y_i \in \mathbb{Z}$, $x_i \leq x_{i+1}$, $y_i \leq y_{i+1}$. 5 can be rewritten as follows:

$$5 = 2 \times 2 + 1 = (2+1) \times 2 - 1 = (1+1) \times 3 - 1.$$

We need $C[2]$ ($v_2 = 2$, $g[v_2] = 0$), $C[3]$ ($v_2 = 2$, $g[v_2] = 1$), and $C[2]$ ($v_3 = 1$, $g[v_3] = 1$).

It is easy to see that the optimal chain

$$C[2] = \langle\langle 1 \rangle, \langle 1 \rangle, \langle 0 \rangle\rangle,$$

and

$$C[3] = \langle\langle 1 \rangle, \langle 0 \rangle, \langle 1 \rangle\rangle.$$

$P(C[2]) = P(C[3]) = 1$.

We choose the best choice among $5 = 2 \times 2 + 1$, $5 = 3 \times 2 - 1$, $5 = 2 \times 3 - 1$. By the first choice, we get the chain

$$C'[5] = \langle\langle 1, 1 \rangle, \langle 0, 0 \rangle, \langle 0, 2 \rangle\rangle.$$

The second choice and the third choice is

$$C''[5] = \langle\langle -1, 1 \rangle, \langle 0, 1 \rangle, \langle 0, 1 \rangle\rangle.$$

We get $P(C'[5]) = P(C''[5]) = 3$, and both of them can be the optimal chain ($C[5] = C'[5] = C''[5]$).

Using the idea explained above, we propose Algorithm 7,8. Most parts of the algorithms are similar to Algorithm 5,6. We only need to compute $v + g[v]$ in addition to v in Algorithm 5,6. We also explain the algorithm using Example 4.

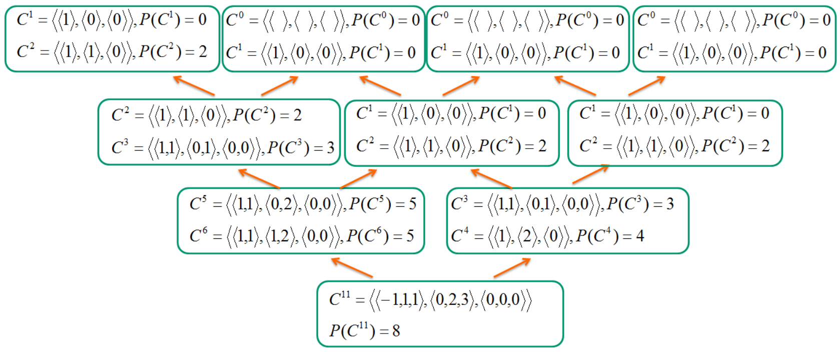

Fig. 5. Calculating $C[11]$ given $Ds = \{0, \pm 1\}$, $P_{add} = 1$, $P_{dou} = 2$, $P_{tri} = 3$. In this case, $G = \{0, 1\}$.

Example 4. Given $Ds = \{0, \pm 1\}$. Find the optimal expansion of $r = 11$ when $P_{tri} = 3$, $P_{dou} = 2$, and $P_{add} = 1$.

In this example, q is initialized to $\lfloor \lg r \rfloor = 3$, and $G = \{0, 1\}$.

- In the first step when $q = 3$, we need to find the optimal expansion of the number with the standard $\left\lfloor \frac{r}{2^3 3^0} \right\rfloor = 1$ and
$$\left\lfloor \frac{r}{2^2 3^1} \right\rfloor = \left\lfloor \frac{r}{2^1 3^2} \right\rfloor = \left\lfloor \frac{r}{2^0 3^3} \right\rfloor = 0.$$
That is $C[0] = C[0+0]$, $C[1] = C[0+1] = C[1+0]$, $C[2] = C[1+1]$. By Line 7-10 of Algorithm 7, $C[0] = \langle \langle \rangle, \langle \rangle, \langle \rangle \rangle$, and $C[1] = \langle \langle 1 \rangle, \langle 0 \rangle, \langle 0 \rangle \rangle$. To find the value of $C[2]$, we get $v_2 = 1$, and $v_3 = 0$. We note that the case we are considering is special as $C[2]$ is one of the input of Algorithm 8 ($2 \in 1 + G$). However, we do not need $C[2]$ to compute itself, and Algorithm 8 can be terminated without the input $C[2]$.
We find $c_{2,u}$ for each u in Lines 1-11 of Algorithm 8. When $u \in \{\pm 1\}$, $va - u \equiv 1 \bmod 2$, and $c_{2,u} \leftarrow \infty$. When $u = 0$, $c_{2,0} = P(C[1]) + P_{dou} = 0 + 2 = 2$. Then, $c_2 \leftarrow 2$, $u_2 \leftarrow 0$, and $vc_2 \leftarrow 1$.
We find $c_{3,u}$ for each u in Lines 12-22 of Algorithm 8. $va - u \equiv 1 \bmod 3$ when $u = 1$, and $va - u \equiv 2 \bmod 3$ when $u = 0$. Then, $c_{3,u} \leftarrow \infty$ when $u \in \{0, 1\}$. When $u = -1$,
$$va - u = 3 \equiv 0 \bmod 3,$$
and
$$c_{3,-1} = P(C^1) + P_{tri} + P_{add} = 0 + 3 + 1 = 4.$$
Then, $c_3 \leftarrow 4$, $u_3 \leftarrow -1$, and
$$vc_3 = \frac{2 - (-1)}{3} = 1.$$
Since $c_2 \leq c_3$, $C[2] \leftarrow \langle R[2], X[2], Y[2] \rangle$ where $R[2] = \langle 1 \rangle$, $X[2] = \langle 1 \rangle$, $Y[2] = \langle 0 \rangle$ by Line 24 of Algorithm 8.
- Next, we consider the case when $q = 2$. That is to find the optimal expansion of the number with the standard
$$\left\lfloor \frac{11}{2^2 3^0} \right\rfloor = 2, \left\lfloor \frac{11}{2^1 3^1} \right\rfloor = \left\lfloor \frac{11}{2^0 3^2} \right\rfloor = 1.$$
That is $C[1] = C[1+0]$, $C[2] = C[1+1] = C[2+0]$, and $C[3] = C[2+1]$. We have already computed $C[1]$ and $C[2]$ in the first step. To compute $C[3]$ ($va = 3$), we get $v_2 = v_3 = 1$.
In Lines 1-11 of Algorithm 8, we find $c_{2,u}$. When $u = -1$, $\frac{va-u}{2} = 2$. From the result on the previous step, $P(C[2]) = 2$, and
$$c_{2,-1} = 2 + P_{dou} + P_{add} = 2 + 2 + 1 = 5.$$
When $u = 0, va - u = 1 \equiv 1 \mod 2$. Then, $c_{2,0} = \infty$. When $u = 1$, $\frac{va-u}{2} = 1$. As $P(C[1]) = 0$,
$$c_{2,1} = 0 + P_{dou} + P_{add} = 2 + 1 = 3.$$

Hence, $c_2 \leftarrow 3$, $u_2 \leftarrow 1$, $vc_2 \leftarrow 1$.
In Lines 12-22 of Algorithm 8, we find $c_{3,u}$. When $u \in \{\pm 1\}$, $va - u \not\equiv 0 \bmod 3$, and $c_{3,u} \leftarrow \infty$. When $u = 0$,

$$c_{3,0} = P(C^1) + P_{tri} = 0 + 3 = 3.$$

Since $c_2 \le c_3$, $C[3] \leftarrow \langle R[3], X[3], Y[3] \rangle$ where $R[3] = \langle 1, 1 \rangle$, $X[3] = \langle 0, 1 \rangle$, $Y[3] = \langle 0, 0 \rangle$ by Line 26 of Algorithm 8.

– When $q = 1$, we find the optimal expansion of the number with the standard $\left\lfloor \frac{11}{2^1 3^0} \right\rfloor = 5$ and $\left\lfloor \frac{11}{2^0 3^1} \right\rfloor = 3$. Those are $C[3] = C[3+0]$, $C[4] = C[3+1]$, $C[5] = C[5+0]$, and $C[6] = C[5+1]$.
We have already computed $C[3]$ on the previous step. Then, we proceed to $C[4]$. In this case,

$$c_2 \leftarrow c_{2,0} = P(C[2]) + P_{dou} = 2 + 2 = 4,$$

$u_2 \leftarrow 0$, and

$$vc_2 \leftarrow \frac{4-0}{2} = 2.$$

$$c_3 \leftarrow c_{1,1} = P(C^1) + P_{tri} + P_{add} = 0 + 3 + 1 = 4.$$

As a result, we use Line 24 of Algorithm 8 to edit

$$C[2] = \langle \langle 1 \rangle, \langle 1 \rangle, \langle 0 \rangle \rangle$$

to

$$C[4] = \langle \langle 1 \rangle, \langle 2 \rangle, \langle 0 \rangle \rangle.$$

For $C[5]$,

$$c_2 \leftarrow c_{2,1} = P(C[2]) + P_{dou} + P_{add} = 2 + 2 + 1 = 5,$$

$u_2 \leftarrow 1$, and

$$vc_2 \leftarrow \frac{5-1}{2} = 2.$$

$$c_3 \leftarrow c_{3,-1} = P(C[2]) + P_{tri} + P_{add} = 2 + 3 + 1 = 6.$$

Then, we use Line 26 of Algorithm 8, and

$$C[5] = \langle \langle 1, 1 \rangle, \langle 0, 2 \rangle, \langle 0, 0 \rangle \rangle.$$

For $C[6]$,

$$c_2 \leftarrow c_{2,0} = P(C[3]) + P_{dou} = 3 + 2 = 5,$$

$u_2 \leftarrow 0$, and $vc_2 \leftarrow 3$.

$$c_3 \leftarrow c_{3,0} = P(C[2]) + P_{tri} = 2 + 3 = 5.$$

Then, we use Line 26 of Algorithm 8, and

$$C[6] = \langle \langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 0, 0 \rangle \rangle.$$

– On the last step, $q = 0$, and we ready to find the solution $C[11]$ in this step.

$$c_2 \leftarrow c_{2,-1} = P(C[6]) + P_{dou} + P_{add} = 5 + 2 + 1 = 8,$$

while

$$c_3 \leftarrow c_{4,-1} = P(C[4]) + P_{tri} + P_{add} = 4 + 3 + 1 = 8.$$

Since $u_2 \leftarrow -1$ and $vc_2 \leftarrow 6$. The solution

$$C[11] \leftarrow \langle\langle -1, 1, 1\rangle, \langle 0, 2, 3\rangle, \langle 0, 0, 0\rangle\rangle.$$

We illustrate this example in Figure 5. Also, we note that when $P_{tri} > P_{dou}$, we rarely use point triples (as we have not used any point triples to compute $C[11]$). The situation is worse when $P_{tri} \approx 2 \times P_{dou}$ in many elliptic curves.

3.3 Generalized Algorithm for Multiple Integers

In this subsection, we expand the algorithm proposed on the previous subsection for the multi-scalar multiplication. In this case, we can use Shamir's trick to reduce the computation cost. The cost in this case $PJ(C)$ when $C = \langle R_1, \ldots, R_d, X, Y\rangle$ is

$$PJ(C) = x_{m-1} \cdot P_{dou} + y_{m-1} \cdot P_{tri} + (JW(C) - 1) \cdot P_{add},$$

as explained in Section 2.

We show the method in Algorithm 9,10. The generalization from Algorithm 7,8 are quite simple. We just replace the single integer with a tuple, and change the cost function P to the joint cost function PJ.

4 Results

To evaluate our algorithm, we show some experimental results in this section. We perform the experiment on each implementation environment such as the scalar multiplication defined on the binary field ($\mathbb{F}_{2^q}$), the scalar multiplication defined on the prime field ($\mathbb{F}_p$), and the multi-scalar multiplication defined on the prime field. To compute point addition, point double, and point triple defined in Section 1, we need to compute field inversion, field squaring, and field multiplication. We define the cost for field inversion as $[i]$, field squaring as $[s]$, and field multiplication as $[m]$. Basically, $P_{dou} = P_{add} = [i] + [s] + 2[m]$. However, there are many researches working on optimizing more complicated operation such as point triple, point quadruple $[3, 4, 19, 20]$. Moreover, when point addition is chosen to perform just after the point double, we can use some intermediate results of point double to reduce the computation time of point addition. Then, it is more convenient to consider point double and point addition together as the basic operation. We call the operation as point double-and-add. The computation cost of point double-and-add is $P_{dou+add}$. The similar thing also happen when we perform point addition after point triple, and we also define point triple-and-add as another basic operation. Also, we define the computation cost of point triple-and-add as $P_{tri+add}$. It is obvious that we can treat these improvements by a little modification in Algorithm 9,10.

4.1 Scalar Multiplication on the Binary Field

In the binary field, the field squaring is very fast, i.e. $[s] \approx 0$. Normally, $3 \leq [i]/[m] \leq 10$. There are two methods to compute point double-and-add, point triple, and point triple-and-add proposed by [19, 20]. In [20],

$$P_{dou+add} = 2[i] + 2[s] + 3[m]$$

while

$$P_{dou+add} = 1[i] + 2[s] + 9[m]$$

in [19]. Then, it is faster to use the method proposed in [19] if $[i]/[m] \leq 6$. Also,

$$P_{tri} = 2[i] + 2[s] + 3[m]$$

in [20], and

$$P_{tri} = 1[i] + 4[s] + 7[m]$$

in [19], and [19] is better when $[i]/[m] \leq 4$. For point triple-and-addition,

$$P_{tri+add} = 3[i] + 3[s] + 4[m]$$

in [20], and

$$P_{tri+add} = 2[i] + 3[s] + 9[m]$$

in [19]. Again, [19] is better when $[i]/[m] \leq 5$.

In this subsection, we use the same parameter as [3] does, and perform two experiments. First, We set $[i]/[m] = 4$ and use the method from [20]. In this case,

$$P_{dou} = P_{add} = [i] + [s] + 2[m] = 6[m],$$

$$P_{dou+add} = P_{tri} = 2[i] + 2[s] + 3[m] = 11[m],$$

and

$$P_{tri+add} = 3[i] + 3[s] + 4[m] = 16[m].$$

In another experiment, we set $[i]/[m] = 8$ and use the method from [19]. In this case,

$$P_{dou} = P_{add} = [i] + [s] + 2[m] = 10[m],$$

$$P_{dou+add} = 1[i] + 2[s] + 9[m] = 17[m],$$

$$P_{tri} = 1[i] + 4[s] + 7[m] = 15[m],$$

and

$$P_{tri+add} = 2[i] + 3[s] + 9[m] = 25[m].$$

In both experiments, we set $Ds = \{0, \pm 1\}$, and we randomly select 10000 positive integers which are less than 2^{163}, and find the average computation cost comparing between the optimal chain proposed in this paper and the greedy algorithm presented in [3, 4]. The results are shown in Table 1. Our result is 4.06% better than [3] when $[i]/[m] = 4$, and 4.77% better than [3] when $[i]/[m] = 8$.

Table 1. Comparing the computation cost for scalar point multiplication using double-base chains when the elliptic curve is implemented in the binary field

Method	$[i]/[m] = 4$	$[i]/[m] = 8$
Binary	$1627[m]$	$2441[m]$
NAF [1]	$1465[m]$	$2225[m]$
Ternary/Binary [12]	$1463[m]$	$2168[m]$
DB-Chain (Greedy) [3]	$1427[m]$	$2139[m]$
Optimized DB-Chain (Our Result)	$1369[m]$	$2037[m]$

4.2 Scalar Multiplication on the Prime Field

When we compute the scalar multiplication on the prime field, field inversion is very expensive task as $[i]/[m]$ is usually more than 30. To cope with that, we compute in the coordinate in which we need to perform field inversion as least as possible such as inverted Edward coordinate with a curve in Edwards form [21]. Up to this state, it is the fastest way to implement scalar multiplication. In this case, the number of field inversion required in each scalar multiplication is constant, and $P_{add} = 9[m]+1[s]$, $P_{dou} = 3[m]+4[s]$, and $P_{tri} = 9[m]+4[s]$ [22]. To compare our results with the existing work, we set the parameter similar to what [6] does. $[s] = 0.8[m]$, and $Ds = \{0, \pm 1\}$. We perform five experiments, for the positive integer less than 2^{256}, 2^{320}, 2^{384}, 2^{448}, and 2^{512}. In each experiment, we randomly select 10000 integer, and find the average computation cost in term of $[m]$. We show the results in Table 2. Our results improve the tree-based approach proposed by Doche and Habsieger by $3.95\%, 3.88\%, 3.90\%, 3.90\%, 3.90\%$ when the bit number is 192 bits, 256 bits, 320 bits, 384 bits, 512 bits respectively.

Table 2. Comparing the computation cost for scalar point multiplication using double-base chains when the elliptic curve is implemented in the prime field

Method	192 bits	256 bits	320 bits	384 bits	512 bits
NAF [1]	$1817.6[m]$	$2423.5[m]$	$3029.3[m]$	$3635.2[m]$	$4241.1[m]$
Ternary/Binary [12]	$1761.2[m]$	$2353.6[m]$	$2944.9[m]$	$3537.2[m]$	$4129.6[m]$
DB-Chain (Greedy) [4]	$1725.5[m]$	$2302.0[m]$	$2879.1[m]$	$3455.2[m]$	$4032.4[m]$
Tree-Based Approach [6]	$1691.3[m]$	$2255.8[m]$	$2821.0[m]$	$3386.0[m]$	$3950.3[m]$
Our Result	$1624.5[m]$	$2168.2[m]$	$2710.9[m]$	$3254.1[m]$	$3796.3[m]$

We also compare our results with the other digit sets. In this case, we compare our results with the works by Bernstein et al. [5]. In the paper, they use the different way to measure the computation cost of the scalar multiplication. In addition to the cost of computing rS, they also consider the cost for the precomputation. For example, we need to precompute $3S, 5S, \ldots, 17S$ when $Ds = \{0, \pm 1, \pm 3, \ldots, \pm 17\}$. We perform the experiment on eight different curves and coordinates. In each curve, the computation cost for point double, point addition, and point triple are different, and we use the same parameter as defined in [5]. We use $Ds = \{0, \pm 1, \pm 3, \ldots, \pm(2h+1)\}$, and we check all $0 \leq h \leq 20$ to find the digit set that give us the minimal average computation cost. Although, the computation cost of the scalar multiplication tends to be lower if we use larger digit set, the higher precomputation cost makes optimal h lied between 6 to 8 in most of cases.

Recently, there is a research by Meloni and Hasan [13]. Instead of using double-base chain, they use double-base number system defined in Section 1. To cope with the difficulties computing the number system, they introduce Yao's algorithm. Their result significantly improves the result using the double-base chain using greedy algorithm, especially the curve where point triple is expensive.

In Table 3-4, we compare the results in [5], [13] with our algorithm. In Table 4, we randomly choose 10000 positive integers less than 2^{160} while the numbers are less than 2^{256}. As the improvement from [5], our algorithm significantly improves the efficiency. On the other hand, our results do not improve the result from [13] in many cases. These cases are the case when point triple is costly operation, and we need only few point triples in the optimal chain. In this case, Yao's algorithm works efficiently. However, our algorithm works better in the inverted Edward coordinate, which is commonly used as the benchmark to compare the algorithm.

Table 3. Comparing the computation cost for scalar point multiplication using double-base chains in larger digit set when the elliptic curve is implemented in the prime field, and the bit number is 160

Method	3DIK	Edwards	ExtJQuartic	Hessian
DBC + Greedy Alg. [5]	1502.4[m]	1322.9[m]	1311.0[m]	1565.0[m]
DBNS + Yao's Alg. [13]	1477.3[m]	1283.3[m]	1226.0[m]	1501.8[m]
Our Algorithm	1438.7[m]	1284.3[m]	1276.5[m]	1514.4[m]

Method	InvEdwards	JacIntersect	Jacobian	Jacobian-3
DBC + Greedy Alg. [5]	1290.3[m]	1438.8[m]	1558.4[m]	1504.3[m]
DBNS + Yao's Alg. [13]	1258.6[m]	1301.2[m]	1534.9[m]	1475.3[m]
Our Algorithm	1257.5[m]	1376.0[m]	1514.5[m]	1458.0[m]

4.3 Multi-scalar Multiplication on the Prime Field

In this subsection, we use the algorithm proposed in Subsection 3.3, and compare our experimental results with the work by Doche et al. [14]. In this experiment, we are interested on the scalar multiplication when $d = 2$, i.e. we have two inputs r_1, r_2. We set $Ds = \{0, \pm 1\}$, and use inverted Edwards coordinates. There are

Table 4. Comparing the computation cost for scalar point multiplication using double-base chains in larger digit set when the elliptic curve is implemented in the prime field, and the bit number is 256. The results in this table is different from the others. Each number is the cost for computing a scalar multiplication added by the precomputation time. In each case, we find the digit set Ds that makes the number minimal.

Method	3DIK	Edwards	ExtJQuartic	Hessian
DBC + Greedy Alg. [5]	2393.2[m]	2089.7[m]	2071.2[m]	2470.6[m]
DBNS + Yao's Alg. [13]	2319.2[m]	2029.8[m]	1991.4[m]	2374.0[m]
Our Algorithm	2287.4[m]	2031.2[m]	2019.4[m]	2407.4[m]

Method	InvEdwards	JacIntersect	Jacobian	Jacobian-3
DBC + Greedy Alg. [5]	2041.2[m]	2266.1[m]	2466.2[m]	2379.0[m]
DBNS + Yao's Alg. [13]	1993.3[m]	2050.0[m]	2416.2[m]	2316.2[m]
Our Algorithm	1989.9[m]	2173.5[m]	2413.2[m]	2319.9[m]

six experiments shown in Table 5. In each experiment, we randomly select 10000 pairs of positive integers, which are less than 2^{192}, 2^{256}, 2^{384}, 2^{448}, 2^{512}. Our algorithm improves the tree-based approach by 2.81%, 2.69%, 2.55%, 3.58%, 3.80%, 4.37% when the bit number is 192,256,320,384,448,512 respectively.

Table 5. Comparing the computation cost for multi scalar multiplication using double-base chains when the elliptic curve is implemented in the prime field

Method	192 bits	256 bits	320 bits	384 bits	448 bits	512 bits
JSF [15]	2044[m]	2722[m]	3401[m]	4104[m]	4818[m]	5531[m]
JBT [14]	2004[m]	2668[m]	3331[m]	4037[m]	4724[m]	5417[m]
Tree-Based [14]	1953[m]	2602[m]	3248[m]	3938[m]	4605[m]	5292[m]
Our Result	1898[m]	2532[m]	3165[m]	3797[m]	4430[m]	5061[m]

We also compare our algorithm in the case that digit set is larger than $\langle 0, \pm 1\rangle$. In [14], there is also a result when $Ds = \{0, \pm 1, \pm 5\}$. Then, we compare our result with the result in Table 6. In this case, our algorithm improves the tree-based approach by 4.1%, 3.7%, 3.5%, 4.6%, 4.6%, 5.1% when the bit number is 192,256,320,384,448,512 respectively.

Moreover, we observe that the hybrid binary-ternary number system proposed by Adikari et al. [16, 17] is the double-base chains for multi-scalar multiplication when $Ds = \{0, \pm 1, \pm 2, 3\}$. Although the conversion algorithm is comparatively fast, there is a large gap between the efficiency of their outputs and the optimal output. We show in Table 7 that the computation cost of the optimal chains are better than the hybrid binary-ternary number system by $11 - 12\%$ on average.

Table 6. Comparing the computation cost for multi scalar multiplication using double-base chains when the elliptic curve is implemented in the prime field, and $Ds = \{0, \pm 1, \pm 5\}$

Method	192 bits	256 bits	320 bits	384 bits	448 bits	512 bits
Tree-Based [14]	1795[m]	2390[m]	2984[m]	3624[m]	4234[m]	4862[m]
Our Result	1722[m]	2301[m]	2880[m]	3457[m]	4039[m]	4616[m]

Table 7. Comparing the computation cost for multi scalar multiplication using double-base chains when the elliptic curve is implemented in the prime field, and $Ds = \{0, \pm 1, \pm 2, 3\}$

Method	192 bits	256 bits	320 bits	384 bits	448 bits	512 bits
Hybrid Binary-Ternary [16, 17]	1905[m]	2537[m]	3168[m]	3843[m]	4492[m]	5159[m]
Our Result	1694[m]	2263[m]	2832[m]	3400[m]	3968[m]	4537[m]

5 Conclusion and Future Works

In this work, we use the dynamic programming algorithm to present the optimal double-base chain. The chain guarantees the optimal computation cost on the scalar multiplication. The time complexity of the algorithm is $O(\lg^2 r)$, similar to the greedy algorithm. Also, our algorithms consume the same amount of memory as the tree-based approach, $O(\lg^2 r)$. The experiment result shows that the optimal chain significantly improve the efficiency of scalar multiplication from the greedy algorithm.

As a future works, we want to analyze the minimal average number of term required for each integer. It is proved that the average number of term required to define integer r, when $0 \leq r < 2^q$ is $o(q)$ [11, 12]. Even the output of the greedy algorithm, the number of term is $o(q)$. However, the greedy algorithm need $\Theta(q)$ for the double-base chain. Then, it is interesting to prove whether the average optimal number of term in the chain is $o(q)$. The result might introduce us to sub-linear time scalar multiplication.

Another future work is to apply the dynamic programming algorithm to DBNS. As [13] introduce Yao's algorithm on the greedy result to improve the operation, the optimal result can reduce the computation cost further. However, there are some clues in our recent results pointed out that the problem might be NP-hard.

References

1. Egecioglu, O., Koc, C.K.: Exponentiation using canonical recoding. Theoretical Computer Science **8**(1) (1994) 19–38
2. Muir, J.A., Stinson, D.R.: New minimal weight representation for left-to-right window methods. Department of Combinatorics and Optimization, School of Computer Science, University of Waterloo (2004)

3. Dimitrov, V., Imbert, L., Mishra, P.K.: Efficient and secure elliptic curve point multiplication using double-base chains. In: Proc. of ASIACRYPT 2005. (2005) 59–78
4. Dimitrov, V., Imbert, L., Mishra, P.K.: The double-base number system and its application to elliptic curve cryptography. Mathematics of Computation **77** (2008) 1075–1104
5. Bernstein, D.J., Birkner, P., Lange, T., Peters, C.: Optimizing double-base elliptic-curve single-scalar multiplication. In: In Progress in Cryptology - INDOCRYPT 2007. Volume 4859 of Lecture Notes in Computer Science., Springer (2007) 167–182
6. Doche, C., Habsieger, L.: A tree-based approach for computing double-base chains. In: ACISP 2008. (2008) 433–446
7. Suppakitpaisarn, V.: Optimal average joint hamming weight and digit set expansion on integer pairs. Master's thesis, The University of Tokyo (2009)
8. Suppakitpaisarn, V., Edahiro, M.: Fast scalar-point multiplication using enlarged digit set on integer pairs. Proc. of SCIS 2009 (2009) 14
9. Suppakitpaisarn, V., Edahiro, M., Imai, H.: Optimal average joint hamming weight and minimal weight conversion of d integers. Cryptology ePrint Archive **2010/300** (2010)
10. Imbert, L., Philippe, F.: How to compute shortest double-base chains? In: ANTS IX. (July 2010)
11. Dimitrov, V., Cooklev, T.V.: Two algorithms for modular exponentiation based on nonstandard arithmetics. IEICE Trans. Fundamentals **E78-A**(1) (January 1995) 82–87 special issue on cryptography and information security.
12. Dimitrov, V.S., Jullien, G.A., Miller, W.C.: An algorithm for modular exponentiations. Information Processing Letters **66** (1998) 155–159
13. Meloni, N., Hasan, M.A.: Elliptic curve scalar multiplication combining yao's algorithm and double bases. In: CHES 2009. (2009) 304–316
14. Doche, C., Kohel, D.R., Sica, F.: Double-base number system for multi-scalar multiplications. In: EUROCRYPT 2009. (2009) 502–517
15. Solinas, J.A.: Low-weight binary representation for pairs of integers. Centre for Applied Cryptographic Research, University of Waterloo, Combinatorics and Optimization Research Report CORR (2001)
16. Adikari, J., Dimitrov, V.S., Imbert, L.: Hybrid binary-ternary number system for elliptic curve cryptosystems. In: ARITH19. (2009) 76–82
17. Adikari, J., Dimitrov, V., Imbert, L.: Hybrid binary-ternary number system for elliptic curve cryptosystems. IEEE Transactions on Computers **99** (2010) to appear
18. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. on Information Theory **IT-31** (1985) 469–472
19. Ciet, M., Joye, M., Lauter, K., Montgomery, P.L.: Trading inversions for multiplications in elliptic curve cryptography. Designs, Codes and Cryptography **39**(6) (2006) 189–206
20. Eisentrager, K., Lauter, K., Montgomery, P.L.: Fast elliptic curve arithmetic and improved Weil pairing evaluation. In: Topics in Cryptology - CT-RSA 2003. Volume 2612 of Lecture Notes in Computer Science., Springer (2003) 343–354
21. Bernstein, D.J., Lange, T.: Inverted edwards coordinates. In Boztas, S., H.-F.Lu, eds.: AAECC 2007. Volume 4851 of Lecture Note in Computer Science., Heidelberg, Springer (2007) 20–27
22. Bernstein, D., Lange, T.: Explicit-formulas database

Appendix A: The Carry Set

In this section, we present the algorithm to find the carry set G. We show the method in Algorithm 11. It is based on breadth-first search scheme. And, we find the upper bound of the cardinality of the carry set in Lemma 1.

Lemma 1. *Given the finite digit set Ds, Algorithm 3 always terminates. And,*

$$||G|| \leq \max Ds - \min Ds + 2,$$

when G is the output carry set.

Proof. Since

$$\begin{aligned} G = &\{\frac{c-d}{2} \in \mathbb{Z} | d \in Ds \wedge c \in G\} \cup \{\frac{c-d+1}{2} \in \mathbb{Z} | d \in Ds \wedge c \in G\} \\ &\cup \{\frac{c-d}{2} \in \mathbb{Z} | d \in Ds \wedge c \in G\} \cup \{\frac{c-d+1}{2} \in \mathbb{Z} | d \in Ds \wedge c \in G\}. \end{aligned}$$

$$\min G \geq \frac{\min G - \max Ds}{2}.$$

Then,

$$\min G \geq -\max Ds.$$

Also,

$$\max G \leq -\min Ds + 1.$$

We conclude that if Ds is finite, G is also finite. And, Algorithm 3 always terminates.

$$||G|| \leq \max Ds - \min Ds + 2.$$

$\square$

Algorithm 6 $FindOptimal(v, C[v_2], C[v_3])$

Require: the positive integer v,
the optimal double base chain of $\left\lfloor \frac{v}{2} \right\rfloor$, $C[v_2]$,
and the optimal double base chain of $\left\lfloor \frac{v}{3} \right\rfloor$, $C[v_3]$
Ensure: the optimal double base chain of v, $C[v]$

$c_2 \leftarrow P(C[v_2]) + P_{dou}$
if $v \equiv 1 \bmod 2$ **then**
  $c_2 \leftarrow c_2 + P_{add}$
end if
$c_3 \leftarrow P(C[v_3]) + P_{tri}$
if $v \equiv 1 \bmod 3$ **then**
  $c_3 \leftarrow c_3 + P_{add}$
else if $v \equiv 2 \bmod 3$ **then**
  $c_3 \leftarrow \infty$
end if
if $c_2 \leq c_3$ and $v \equiv 0 \bmod 2$ **then**
  $C[v] \leftarrow \langle R[v], X[v], Y[v] \rangle$ where
  $R[v] = R[v_2]$
  $X[v] = \langle x[v]_0, \ldots, x[v]_{m-1} \rangle$ where $x[v]_t \leftarrow x[v_2]_t + 1$
  $Y[v] = Y[v_2]$
else if $c_2 \leq c_3$ and $v \equiv 1 \bmod 2$ **then**
  $C[v] \leftarrow \langle R[v], X[v], Y[v] \rangle$ where
  $R[v] = \langle 1, R[v_2] \rangle$
  $X[v] = \langle 0, x[v]_1, \ldots, x[v]_{m-1} \rangle$ where $x[v]_t \leftarrow x[v_2]_{t-1} + 1$
  $Y[v] = \langle 0, Y[v_2] \rangle$
else if $v \equiv 0 \bmod 3$ **then**
  $C[v] \leftarrow \langle R[v], X[v], Y[v] \rangle$ where
  $R[v] = R[v_3]$
  $X[v] = X[v_3]$
  $Y[v] = \langle y[v]_0, \ldots, y[v]_{m-1} \rangle$ where $y[v]_t \leftarrow y[v_3]_t + 1$
else if $v \equiv 1 \bmod 3$ **then**
  $C[v] \leftarrow \langle R[v], X[v], Y[v] \rangle$ where
  $R[v] = \langle 1, R[v_3] \rangle$
  $X[v] = \langle 0, X[v_3] \rangle$
  $Y[v] = \langle 0, y[v]_1, \ldots, y[v]_{m-1} \rangle$ where $y[v]_t \leftarrow y[v_3]_{t-1} + 1$
end if

Algorithm 7 The algorithm finding the optimal double-base chain for single integer for any Ds

Require: the positive integer r, the finite digit set Ds, and the carry set G
Ensure: the optimal double-base chain of r, $C[r] = \langle R[r], X[r], Y[r] \rangle$

$q \leftarrow \lfloor \lg r \rfloor$
while $q \geq 0$ **do**
  for all $x, y \in \mathbb{Z}^+$ such that $x + y = q$ **do**
    $v \leftarrow \left\lfloor \frac{r}{2^x 3^y} \right\rfloor$
    for all $g[v] \in G$ **do**
      $va \leftarrow v + g[v]$
      if $va = 0$ **then**
        $C[va] \leftarrow \langle \langle \rangle, \langle \rangle, \langle \rangle \rangle$
      else if $va \in Ds$ **then**
        $C[va] \leftarrow \langle \langle va \rangle, \langle 0 \rangle, \langle 0 \rangle \rangle$
      else
        $v_2 \leftarrow \left\lfloor \frac{r}{2^{x+1} 3^y} \right\rfloor$
        $v_3 \leftarrow \left\lfloor \frac{r}{2^x 3^{y+1}} \right\rfloor$
        $C[va] \leftarrow FindOptimal(va, C[v_2 + G], C[v_3 + G])$
      end if
    end for
  end for
  $q \leftarrow q - 1$
end while

Algorithm 8 $FindOptimal(va, C[v_2 + G], C[v_3 + G])$

Require: the positive integer va,
the optimal double base chain of $\lfloor \frac{va}{2} \rfloor + g[\lfloor \frac{va}{2} \rfloor]$ for all $g[\lfloor \frac{va}{2} \rfloor] \in G$, $C[va_2 + G]$,
and the optimal double base chain of $\lfloor \frac{va}{3} \rfloor + g[\lfloor \frac{va}{3} \rfloor]$ for all $g[\lfloor \frac{va}{3} \rfloor] \in G$, $C[va_3 + G]$

Ensure: the optimal double base chain of va, $C[va]$

for all $u \in Ds$ **do**
    if $va - u \equiv 0 \bmod 2$ **then**
        $c_{2,u} \leftarrow P(C[\frac{va-u}{2}]) + P_{dou}$
        if $u \neq 0$ **then**
            $c_{2,u} \leftarrow c_{2,u} + P_{add}$
        else
            $c_{2,u} \leftarrow \infty$
        end if
    end if
end for
$c_2 \leftarrow \min_{u \in Ds} c_{2,u}$, $u_2 \leftarrow \mathrm{minarg}_{u \in Ds} c_{2,u}$, $vc_2 \leftarrow \frac{va-u_2}{2}$
for all $u \in Ds$ **do**
    if $va - u \equiv 0 \bmod 3$ **then**
        $c_{3,u} \leftarrow P(C[\frac{va-u}{3}]) + P_{tri}$
        if $u \neq 0$ **then**
            $c_{3,u} \leftarrow c_{3,u} + P_{add}$
        else
            $c_{3,u} \leftarrow \infty$
        end if
    end if
end for
$c_3 \leftarrow \min_{u \in Ds} c_{3,u}$, $u_3 \leftarrow \mathrm{minarg}_{u \in Ds} c_{3,u}$, $vc_3 \leftarrow \frac{va-u_3}{3}$
if $c_2 \leq c_3$ and $u_2 = 0$ **then**
    $C[v] \leftarrow \langle R[v], X[v], Y[v] \rangle$ where
    $R[v] = R[vc_2]$
    $X[v] = \langle x[v]_0, \ldots, x[v]_{m-1} \rangle$ where $x[v]_t \leftarrow x[vc_2]_t + 1$
    $Y[v] = Y[vc_2]$
else if $c_2 \leq c_3$ **then**
    $C[v] \leftarrow \langle R[v], X[v], Y[v] \rangle$ where
    $R[v] = \langle u_2, R[vc_2] \rangle$
    $X[v] = \langle 0, x[v]_1, \ldots, x[v]_{m-1} \rangle$ where $x[v]_t \leftarrow x[vc_2]_{t-1} + 1$
    $Y[v] = \langle 0, Y[vc_2] \rangle$
else if $u_3 = 0$ **then**
    $C[v] \leftarrow \langle R[v], X[v], Y[v] \rangle$ where
    $R[v] = R[vc_3]$
    $X[v] = X[vc_3]$
    $Y[v] = \langle y[v]_0, \ldots, y[v]_{m-1} \rangle$ where $y[v]_t \leftarrow y[vc_3]_t + 1$
else
    $C[v] \leftarrow \langle R[v], X[v], Y[v] \rangle$ where
    $R[v] = \langle u_3, R[vc_3] \rangle$
    $X[v] = \langle 0, X[vc_3] \rangle$
    $Y[v] = \langle 0, y[v]_1, \ldots, y[v]_{m-1} \rangle$ where $y[v]_t \leftarrow y[vc_3]_{t-1} + 1$
end if

Algorithm 9 The algorithm finding the optimal double-base chain for d integers on any digit set Ds

Require: the positive integer $r_1, \ldots, r_d$, the finite digit set Ds, and the carry set G
Ensure: the optimal double-base chain of $M = \langle r_1, \ldots, r_d \rangle$,
$C[M] = \langle R[M]_1, \ldots, R[M]_d, X[M], Y[M] \rangle$

$q \leftarrow \max_i \lfloor \lg r_i \rfloor$
while $q \geq 0$ **do**
  for all $x, y \in \mathbb{Z}^+$ such that $x + y = q$ **do**
    $v_i \leftarrow \left\lfloor \frac{r_i}{2^x 3^y} \right\rfloor$ for $1 \leq i \leq d$
    for all $\langle g[v]_1 \ldots g[v]_d \rangle \in G^d$ **do**
      $va_i \leftarrow v_i + g_i^v$ for $1 \leq i \leq d$, $V \leftarrow \langle va_1, \ldots, va_d \rangle$
      if $V = \mathbf{0}$ **then**
        $C[V] \leftarrow \langle \langle \rangle, \langle \rangle, \langle \rangle \rangle$
      else if $V \in Ds^d$ **then**
        $C[V] \leftarrow \langle \langle va_1 \rangle, \ldots, \langle va_d \rangle, \langle 0 \rangle, \langle 0 \rangle \rangle$
      else
        $V_2 \leftarrow \langle \left\lfloor \frac{r_1}{2^{x+1} 3^y} \right\rfloor, \ldots, \left\lfloor \frac{r_d}{2^{x+1} 3^y} \right\rfloor \rangle$
        $V_3 \leftarrow \langle \left\lfloor \frac{r_1}{2^x 3^{y+1}} \right\rfloor, \ldots, \left\lfloor \frac{r_d}{2^x 3^{y+1}} \right\rfloor \rangle$
        $C[V] \leftarrow FindOptimal(V, C[V_2 + G[d]], C[V_3 + G[d]])$
      end if
    end for
  end for
  $q \leftarrow q - 1$
end while

Algorithm 10 $FindOptimal(V, C[V_2+G], C[V_3+G])$

Require: $V = \langle va_1, \ldots, va_d \rangle$ where va_i are possitive integers,
the optimal double base chain of $V_2 + g[V_2]$ for all $g[V_2] \in G$, $C[V_2+G]$,
and the optimal double base chain of $V_3 + g[V_3]$ for all $g[V_3] \in G$, $C[V_3+G]$
Ensure: the optimal double base chain of V, $C[V]$

for all $U = \langle u_1 \ldots u_d \rangle \in Ds^d$ **do**
  if $va_i - u_i \equiv 0 \bmod 2$ for all $1 \leq i \leq d$ **then**
    $VC_{2,U} \leftarrow \langle \frac{va_1 - u_1}{2}, \ldots \frac{va_d - u_d}{2} \rangle$
    $c_{2,U} \leftarrow PJ(C[VC_{2,U}]) + P_{dou}$
    if $U \neq \mathbf{0}$ **then**
      $c_{2,U} \leftarrow c_{2,U} + P_{add}$
    else
      $c_{2,U} \leftarrow \infty$
    end if
  end if
end for
$c_2 \leftarrow \min_{U \in Ds^d} c_{2,U}$
$U_2 \leftarrow \mathrm{minarg}_{U \in Ds} c_{2,U}$
$VC_2 \leftarrow VC_{2,U_2}$
for all $U = \langle u_1 \ldots u_d \rangle \in Ds^d$ **do**
  if $va_i - u_i \equiv 0 \bmod 3$ for all $1 \leq i \leq d$ **then**
    $VC_{3,U} \leftarrow \langle \frac{va_1 - u_1}{3}, \ldots \frac{va_d - u_d}{3} \rangle$
    $c_{3,U} \leftarrow PJ(C[VC_{3,U}]) + P_{tri}$
    if $U \neq \mathbf{0}$ **then**
      $c_{3,U} \leftarrow c_{3,U} + P_{add}$
    else
      $c_{3,U} \leftarrow \infty$
    end if
  end if
end for
$c_3 \leftarrow \min_{U \in Ds^d} c_{3,U}$
$U_3 \leftarrow \mathrm{minarg}_{U \in Ds} c_{3,U}$
$VC_3 \leftarrow VC_{3,U_3}$
if $c_2 \leq c_3$ and $U = \mathbf{0}$ **then**
  $C[V] \leftarrow \langle R[V]_1, \ldots, R[V]_d, X[V], Y[V] \rangle$ where
  $R[V]_i = R[VC_2]_i$ for $1 \leq i \leq d$
  $X[V] = \langle x[V]_0, \ldots, x[V]_{m-1} \rangle$ where $x[V]_t \leftarrow x[VC_2]_t + 1$
  $Y[V] = Y[VC_2]$
else if $c_2 \leq c_3$ **then**
  $C[V] \leftarrow \langle R[V]_1, \ldots, R[V]_d, X[V], Y[V] \rangle$ where
  $R[V]_i = \langle U_{2,i}, R[VC_2]_i \rangle$ for $1 \leq i \leq d$
  $X[V] = \langle 0, x[V]_1, \ldots, x[V]_{m-1} \rangle$ where $x[V]_t \leftarrow x[VC_2]_{t-1} + 1$
  $Y[V] = \langle 0, Y[VC_2] \rangle$
else if $U_3 = \mathbf{0}$ **then**
  $C[V] \leftarrow \langle R[V]_1, \ldots, R[V]_d, X[V], Y[V] \rangle$ where
  $R[V]_i = R[VC_3]_i$ for $1 \leq i \leq d$
  $X[V] = X[VC_3]$
  $Y[V] = \langle y[V]_0, \ldots, y[V]_{m-1} \rangle$ where $y[V]_t \leftarrow y[VC_3]_t + 1$
else
  $C[V] \leftarrow \langle R[V]_1, \ldots, R[V]_d, X[V], Y[V] \rangle$ where
  $R[V]_i = \langle U_{3,i}, R[VC_3]_i \rangle$ for $1 \leq i \leq d$
  $X[V] = \langle 0, X[VC_3] \rangle$
  $Y[V] = \langle 0, y[V]_1, \ldots, y[V]_{m-1} \rangle$ where $y[V]_t \leftarrow y[VC_3]_{t-1} + 1$
end if

Algorithm 11 Find the carry set of the given digit set

Require: the digit set Ds
Ensure: the carry set G

1: $Ct \leftarrow \{0\}$, $G \leftarrow \oslash$
2: **while** $Ct \neq \oslash$ **do**
3: let $x \in Ct$
4: $Ct \leftarrow Ct \cup (\{\frac{x+d}{2} \in \mathbb{Z} | d \in Ds\} - G - \{x\})$
5: $Ct \leftarrow Ct \cup (\{\frac{x+d+1}{2} \in \mathbb{Z} | d \in Ds\} - G - \{x\})$
6: $Ct \leftarrow Ct \cup (\{\frac{x+d}{3} \in \mathbb{Z} | d \in Ds\} - G - \{x\})$
7: $Ct \leftarrow Ct \cup (\{\frac{x+d+1}{3} \in \mathbb{Z} | d \in Ds\} - G - \{x\})$
8: $G \leftarrow G \cup \{x\}$
9: $Ct \leftarrow Ct - \{x\}$
10: **end while**