Double-Base Chains for Scalar Multiplications on Elliptic Curves

Wei Yu[1,2], Saud Al Musa[3], and Bao Li[1,4]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China
{yuwei,libao}@iie.ac.cn
[2] Data Assurance and Communications Security Research Center, Chinese Academy of Sciences, Beijing 100093, China
yuwei_1_yw@163.com
[3] College of Computer Science and Engineering, Taibah University, Medina, Saudi Arabia
smusa@taibahu.edu.sa
[4] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. Double-base chains (DBCs) are widely used to speed up scalar multiplications on elliptic curves. We present three results of DBCs. First, we display a structure of the set containing all DBCs and propose an iterative algorithm to compute the number of DBCs for a positive integer. This is the first polynomial time algorithm to compute the number of DBCs for positive integers. Secondly, we present an asymptotic lower bound on average Hamming weights of DBCs $\frac{\log n}{8.25}$ for a positive integer n. This result answers an open question about the Hamming weights of DBCs. Thirdly, we propose a new algorithm to generate an optimal DBC for any positive integer. The time complexity of this algorithm is $\mathcal{O}\left((\log n)^2 \log\log n\right)$ bit operations and the space complexity is $\mathcal{O}\left((\log n)^2\right)$ bits of memory. This algorithm accelerates the recoding procedure by more than 6 times compared to the state-of-the-art Bernstein, Chuengsatiansup, and Lange's work. The Hamming weights of optimal DBCs are over 60% smaller than those of NAFs. Scalar multiplication using our optimal DBC is about 13% faster than that using non-adjacent form on elliptic curves over large prime fields.

Keywords: Elliptic curve cryptography, Scalar multiplication, Double-base chain, Hamming weight

1 Introduction

A double-base chain (DBC), as a particular double-base number system (DBNS) representation, represents an integer n as $\sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i}$ where $c_i \in \{\pm 1\}$, b_i, t_i are non-increasing sequences. It is called an unsigned DBC when $c_i \in \{1\}$. A DBC was first used in elliptic curve cryptography for its sparseness by Dimitrov, Imbert, and Mishra [1], and Ciet, Joye, Lauter, and Montgomery [2]. Scalar multiplication is the core operation in elliptic curve cryptosystems. A DBC allows one to represent an integer in a Horner-like fashion to calculate scalar multiplication such that all

partial results can be reused. In the last decade, DBCs were widely investigated to speed up scalar multiplications [3–5] and pairings [6,7]. The generalizations of DBCs were also applied to the arithmetics of elliptic curves. The generalizations include simultaneously representing a pair of numbers to accelerate multi-scalar multiplications [8–10], using double-base representation to speed up scalar multiplication on Koblitz curves [11], and representing an integer in a multi-base number system to promote scalar multiplications [12–14].

Dimitrov, Imbert, and Mishra pointed out that DBC is highly redundant, and counting the exact number of DBCs is useful to generate optimal DBCs [1]. A precise estimate of the number of unsigned DBNS representation of a given positive integer was presented in [15]. 100 has exactly 402 unsigned DBNS representations and 1000 has 1295579 unsigned DBNS representations. For unsigned DBC, Imbert and Philippe [4] introduced an efficient algorithm to compute the number of unsigned DBCs for a given integer. By their algorithm, 100 has 7 unsigned DBCs and 1000 has 30 unsigned DBCs. DBCs are more redundant than unsigned DBCs. For a given integer n, Doche [16] proposed a recursion algorithm to calculate the number of DBCs with a leading term dividing 2^b3^t. His algorithm is efficient to find the number of DBCs with a leading term dividing 2^b3^t for integers less than 2^{70} and $b,\ t < 70$. But it does not work for calculating the number of DBCs of a positive integer used in elliptic curve cryptography. We will show how to calculate the number of DBCs of a 256−bit integer or even a larger integer.

The Hamming weight is one of the most important factors that affect the efficiency of scalar multiplications. Dimitrov, Imbert, and Mishra proved an asymptotic upper bound $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ on the Hamming weight of DBNS representation by a greedy approach [15]. Every integer n has a DBC with Hamming weight $\mathcal{O}\left(\log n\right)$. The upper bounds of DBNS representations and DBCs have been well investigated, in contrast, the precise lower bounds of DBCs can not be found in any literature. Doche and Habsieger [3] showed that the DBCs produced by the tree approach is shorter than those produced by greedy approach [1] for integers with several hundreds of bits experimentally. They observed that the average Hamming weight of the DBCs produced by the tree approach is $\frac{\log n}{4.6419}$. They also posed an open question that the average Hamming weight of DBCs generated by the greedy approach may be not $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$. We will give affirmation to this question.

Canonic DBCs are the DBCs with the lowest Hamming weight for a positive integer and were introduced by Dimitrov, Imbert, and Mishra [1]. Several algorithms were designed to produce near canonic DBCs such as greedy algorithm [1], binary/ternary approach [2], multi-base non-adjacent form (mbNAF) [13], and tree approach [3]. In Asiacrypt 2014, Doche proposed an algorithm to produce a canonic DBC [16] . As Doche's algorithm was in exponential time, Capuñay and Thériault [7] improved Doche's algorithm to generate a canonic DBC or an optimal DBC. This is the first algorithm to generate an optimal DBC in polynomial time, explicitly $\mathcal{O}\left(\left(\log n\right)^4\right)$ bit operations and $\mathcal{O}\left(\left(\log n\right)^3\right)$ bits of memory. Bernstein, Chuengsatiansup, and Lange [17] presented a directed acyclic graph algorithm (DAG) to produce a canonic DBC or an optimal DBC. Their algorithm takes time $\mathcal{O}\left(\left(\log n\right)^{2.5}\right)$

bit operations and $\mathcal{O}\left((\log n)^{2.5}\right)$ bits of memory. As scalar multiplication requires $\mathcal{O}\left((\log n)^2 \log\log n\right)$ when field multiplications use FFTs, we will focus on producing a canonic DBC or an optimal DBC in the same order of magnitude.

In this paper, we are concerned with the theoretical aspects of DBCs arising from their study to speed up scalar multiplication and producing a canonic DBC or an optimal DBC efficiently. The main contributions are detailed as follows.

1. As Doche's algorithm is in exponential time to compute the number of DBCs with a leading term dividing 2^b3^t [16], we propose an iterative algorithm in $\mathcal{O}\left((\log n)^3\right)$ bit operations and in $\mathcal{O}\left((\log n)^2\right)$ bits of memory. Our algorithm is based on our new structure of the set containing all DBCs. It requires 10 milliseconds for 256-bit integers and 360 milliseconds for 1024-bit integers. Using the iterative algorithm, 100 has 2590 DBCs with a leading term dividing $2^{30}3^4$ and 1000 has 28364 DBCs with a leading term dividing $2^{30}3^6$. These results show that DBCs are redundant. We show that the number of DBCs with a leading term dividing 2^b3^t is the same when $t \geq t_\tau$ for some t_τ. The number of DBCs with a leading term dividing 2^b3^t minus the number of DBCs with a leading term dividing $2^{b_\tau}3^t$ is $(b-b_\tau)\,C_\tau$ when $b \geq b_\tau$ for some b_τ and C_τ. We also present that the number of DBCs with a leading term dividing 2^b3^t is $\mathcal{O}(\log n)$ −bit when both b and t are $\mathcal{O}(\log n)$.
2. Doche and Habsieger posed an open question to decide whether the average Hamming weight of DBCs produced by the greedy approach is $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ or not [3]. We show that an asymptotic lower bound of the average Hamming weight of the DBCs returned by any algorithm for a positive integer n is $\frac{\log n}{8.25}$. This theoretical result answers their open question. Experimental results show that the Hamming weight of canonic DBCs is $0.179822\log n$ for 3000-bit integers. It still has a distance from the theoretical bound.
3. We propose a dynamic programming algorithm to generate an optimal DBC. We introduce an equivalent representative for large integers to improve the efficiency of the dynamic programming algorithm. Our dynamic programming algorithm using equivalent representatives requires $\mathcal{O}\left((\log n)^2 \log\log n\right)$ bit operations and $\mathcal{O}\left((\log n)^2\right)$ bits of memory. It accelerates the recoding procedure by over 6 times compared to Bernstein, Chuengsatiansup, and Lange's algorithm. Many researches [1–3, 6, 7, 16, 17] indicate that the leading term of an optimal DBC is greater than $\frac{n}{2}$ and less than $2n$. We will prove it in this work.
4. Capuñay and Thériault's algorithm [7], Bernstein, Chuengsatiansup, and Lange's DAG algorithm [17], and our algorithms (Algorithms $2-4$) can generate the same optimal DBC for a given integer. Using optimal DBCs to speed up pairing computations has been fully investigated by Capuñay and Thériault's algorithm in [7]. Using optimal DBCs to speed up scalar multiplication on Edwards curves has been studied by Bernstein, Chuengsatiansup, and Lange in [17]. We will study scalar multiplication on Weierstrass curves using optimal DBCs. Over large prime fields, both theoretical analyses and experimental results show that scalar

multiplication protecting against simple side-channel attack using our optimal DBC is about 13% faster than that using NAF.

This paper is organized as follows. In Section 2, we present background of elliptic curves and DBCs. In Section 3, we show the structure of the set containing all DBCs, and give an iterative algorithm to compute the number of DBCs. In Section 4, we show an asymptotic lower bound of the average Hamming weights of DBCs. Section 5 shows a dynamic programming algorithm. Section 6 presents equivalent representatives for large numbers to improve our dynamic programming algorithm and presents the comparisons of several algorithms. Section 7 gives some comparisons of scalar multiplications. Finally, we conclude this work in Section 8.

2 Preliminaries

We give some basics about elliptic curves and DBCs.

2.1 Elliptic Curves

In what follows, point doubling ($2P$), tripling ($3P$), and mixed addition [18] ($P+Q$) are denoted by D, T, and A respectively where P and Q are rational points on an elliptic curve. Cost of scalar multiplications are expressed in terms of field multiplications (**M**) and field squarings (**S**). To allow easy comparisons, we disregard field additions/subtractions and multiplications/divisions by small constants. Moreover, we assume that **S** = 0.8**M** as customary of software implementation (different CPU architectures usually imply different **S** and **M** ration) and that **S** = **M** in the case of implementations on a hardware platform or protecting scalar multiplications against some simple side channel attack by side-channel atomicity [19].

Let $\mathscr{E}_{\mathrm{W}}$ be an elliptic curve over a large prime field $\mathbb{F}_p$ defined by the Weierstrass equation in Jacobian projective coordinate: $Y^2 = X^3 + aXZ^4 + bZ^6$, where $a = -3$, $b \in \mathbb{F}_p$, and $4a^3 + 27b^2 \neq 0$. The respective cost of a doubling, a mixed addition, and a tripling are 3**M**+5**S**, 7**M**+4**S**, and 7**M**+7**S** on $\mathscr{E}_{\mathrm{W}}$ respectively [20, 21]. More about Weierstrass elliptic curves please refer to [22].

The cost of point operations on $\mathscr{E}_{\mathrm{W}}$ are summarized in Table 1. $\mathscr{E}_{\mathrm{W}}$ with **S**= 0.8**M** and $\mathscr{E}_{\mathrm{W}}$ with **S**=**M** are denoted by $\mathscr{E}_{\mathrm{W}}$ 0.8 and $\mathscr{E}_{\mathrm{W}}$ 1 respectively.

Table 1. Cost of elliptic curve point operations

operation	$\mathscr{E}_{\mathrm{W}}$ 0.8	$\mathscr{E}_{\mathrm{W}}$ 1
A	7**M**+4**S**(10.2**M**)	11**M**
D	3**M**+5**S**(7**M**)	8**M**
T	7**M**+7**S**(12.6**M**)	14**M**

2.2 DBCs

DBNS represents an integer as $\sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i}$ where $c_i \in \{\pm 1\}$, and b_i, t_i are non-negative integers. It was first used in elliptic curve cryptography by Dimitrov, Imbert, and Mishra [1]. Meloni and Hasan proposed new algorithms using DBNS

representation to speed up scalar multiplications [23, 24]. The drawback of DBNS representation to compute scalar multiplication is that it requires many precomputations and space to compute scalar multiplication. A DBC is a special case of DBNS representations. It allows us to represent n in a Horner-like fashion such that all partial results can be reused. It is defined as follows.

Definition 1 (DBC [1]) *A DBC represents an integer n as $\sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i}$ where $c_i \in \mathscr{C} = \{\pm 1\}, b_l \geq b_{l-1} \geq \ldots \geq b_1 \geq 0$ and $t_l \geq t_{l-1} \geq \ldots \geq t_1 \geq 0$. We call $2^{b_i} 3^{t_i}$ a term of the DBC, $2^{b_l} 3^{t_l}$ the leading term of the DBC, and l the Hamming weight of the DBC.*

If $\mathscr{C} = \{1\}$, the DBC is called an unsigned DBC. Since computing the negative of a point P can be done virtually at no cost, we usually set $\mathscr{C} = \{\pm 1\}$. The leading term of a DBC encapsulates the total number of point doublings and that of point triplings necessary to compute scalar multiplication nP whose total cost is $(l-1) \cdot A + b_l \cdot D + t_l \cdot T$.

The number 0 has only one DBC that is 0. If a DBC does not exist, we denote it by NULL. We set the Hamming weight of 0 as 0 and that of NULL as a negative integer. A DBC for a negative integer is the negative of the DBC of its absolute value. Therefore, we usually investigate the DBCs of a positive integer.

Some properties of DBCs are useful. Let $n = \sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i}$ be a DBC with $c_i \in \{\pm 1\}, b_l \geq b_{l-1} \geq \ldots \geq b_1$ and $t_l \geq t_{l-1} \geq \ldots \geq t_1$. We have

1. $2^{b_k} 3^{t_k}$ is a factor of $\sum_{i=k}^{l_0} c_i 2^{b_i} 3^{t_i}$, when $k \leq l_0 \leq l$;
2. $\sum_{i=k}^{l_0} c_i 2^{b_i} 3^{t_i}$ is not equal to 0 when $0 < k \leq l_0 \leq l$;
3. $\frac{2^{b_{k+\varsigma}} 3^{t_{k+\varsigma}}}{2^{\varsigma-1}} > \sum_{i=1}^{k} c_i 2^{b_i} 3^{t_i} > -\frac{2^{b_{k+\varsigma}} 3^{t_{k+\varsigma}}}{2^{\varsigma-1}}$, when $1 \leq \varsigma \leq l-k$;
4. $2^{b_l} 3^{t_l} > \frac{n}{2}$ [25];
5. $\sum_{i=1}^{\varsigma} c_i 2^{b_i} 3^{t_i} > 0$ if and only if $c_\varsigma = 1$, when $1 \leq \varsigma \leq l$.

Following from Dimitrov, Imbert, and Mishra's definition of canonic DBC,

Definition 2 (Canonic DBC [15]) *The canonic DBCs of a positive integer n are the ones with minimal Hamming weight.*

The canonic DBCs of a positive integer have the same Hamming weight. When we perform scalar multiplication using a DBC, its Hamming weight is not the only factor affecting the efficiency of scalar multiplication. The cost of point operations should also be considered. The works in [7, 16, 17] indicate the definition of an optimal DBC as follows.

Definition 3 (Optimal DBC) *Let* w *be a DBC of a positive integer n whose leading term is $2^{b_l} 3^{t_l}$ and its Hamming weight is l, and the value function of* w *is defined by* $\mathrm{val}(\mathrm{w}) = (l-1) \cdot A + b_l \cdot D + t_l \cdot T$ *for given numbers $A > 0$, $D \geq 0$, and $T \geq 0$. An optimal DBC of n is the DBC with the smallest value in the set* $\{\mathrm{val}(\mathrm{w}) | \mathrm{w} \in X\}$ *where X is the set containing all DBCs of n.*

Let minL$\{w_1, w_2, \ldots, w_m\}$ be a DBC with the smallest Hamming weight among these DBCs. If the Hamming weight of w is the smallest in a corresponding set, we say w is "minimal". Let minV$\{w_1, w_2, \ldots, w_m\}$ be a DBC with the smallest val(w_i) in the set {val(w_1), val(w_2), ..., val(w_m)}. If more than one DBC has the same Hamming weight or the same value of its value function, we choose the one with the smallest position index i where i is the position index of w_i in the set of $\{w_1, w_2, \ldots, w_m\}$. minL is used to generate canonic DBCs, and minV is used to generate optimal DBCs.

An optimal DBC is associated with an elliptic curve. Let log denote binary logarithm. If the value of $\frac{T}{D}$ is log3, then the optimal DBC is a canonic DBC. In this case, we usually set $D = T = 0$. For canonic DBCs of a positive integer, our concern is their Hamming weight.

3 The Number of DBCs

DBCs are special cases of DBNS representations. In 2008, Dimitrov, Imbert, and Mishra showed an accurate estimate of the number of unsigned DBNS representations for a given positive integer [15]. The number of signed DBNS representation is still an open question.

Dimitrov, Imbert, and Mishra pointed out that counting the exact number of DBCs is useful to show DBC is redundant [1] and to generate an optimal DBC. Dimitrov, Imbert, and Mishra [1] and Imbert and Philippe [4] both noticed that each positive integer has at least one DBC such as binary representation. Imbert and Philippe [4] proposed an elegant algorithm to compute the number of unsigned DBCs for a given integer and presented the first 400 values. These values behave rather irregularly. To determine the precise number of DBCs for a positive integer is usually hard, but we are convinced that this number is infinity. The number of DBCs with a leading term dividing 2^b3^t for a positive integer was first investigated by Doche [16]. His algorithm is very efficient for less than 70−bit integers with a leading term dividing 2^b3^t for the most b and t. The algorithm requires exponential time. Before we present a polynomial time algorithm to calculate the number of DBCs of large integers, a structure of the set containing all DBCs is introduced.

3.1 The Structure of the Set Containing All DBCs

Let $\Phi(b,t,n)$ be the set containing all DBCs of an integer $n \geq 0$ with a leading term strictly dividing 2^b3^t. "Strictly" indicates that the leading term of a DBC $2^{b_l}3^{t_l}$ divides 2^b3^t but is not equal to 2^b3^t. Let $\bar{\Phi}(b,t,n)$ be the set containing all DBCs of an integer $n \leq 0$ with a leading term strictly dividing 2^b3^t. Both definitions of $\Phi(b,t,n)$ and $\bar{\Phi}(b,t,n)$ arise from Imbert and Philippe's structure of unsigned DBCs [4] and Capuñay and Thériault's definition of the set containing all DBCs (see Definition 5 of [7]).

Let z be $2^{b'}3^{t'}$ or $-2^{b'}3^{t'}$ with integers $b' \geq 0$ and $t' \geq 0$. The set $\{w + z |\ w \in \Phi\}$ is denoted by ${}^z\Phi$ (the similar is for $\bar{\Phi}$). ${}^z\Phi$ is inspired by Imbert and Philippe's mark [4]. If $2^b3^t | z$, ${}^z\Phi(b,t,n)$ are the DBCs of $n+z$. Let ${}^{z_1,z_2}\Phi = {}^{z_1}({}^{z_2}\Phi)$. Take $\Phi(1,4,100) = \{3^4+3^3-3^2+1\}$ for example, ${}^{2\cdot3^4}\Phi(1,4,100) = \{2\cdot3^4+3^4+3^3-3^2+1\}$.

Some properties of Φ and $\bar{\Phi}$ are given.

1. If $\Phi = \emptyset$, then ${}^z\Phi = \emptyset$; if $\bar{\Phi} = \emptyset$, then ${}^z\bar{\Phi} = \emptyset$.
2. If $\Phi = \{0\}$, then ${}^z\Phi = \{z\}$; if $\bar{\Phi} = \{0\}$, then ${}^z\bar{\Phi} = \{z\}$.
3. If $n < 0$ or $n \geq 2^b 3^t$ or $b < 0$ or $t < 0$, then $\Phi(b,t,n) = \bar{\Phi}(b,t,-n) = \emptyset$.
4. $\Phi(0,0,0) = \bar{\Phi}(0,0,0) = \{0\}$.
5. A DBC 0 plus z equals to z.
6. A DBC NULL plus z equals to NULL.

Imbert and Philippe's structure of the set containing unsigned DBCs [4] can be used to calculate the number of unsigned DBCs. Since the terms of DBCs of n may be larger than n, calculating the number of DBCs is usually difficult. Following from Capuñay and Thériault's definition [7],

$$n_{b,t} \equiv n \pmod{2^b 3^t} \text{ where } 0 \leq n_{b,t} < 2^b 3^t.$$

We redefine

$$\bar{n}_{b,t} = n_{b,t} - 2^b 3^t.$$

To calculate the number of DBCs, $\Phi(b,t)$ and $\bar{\Phi}(b,t)$ are introduced to describe the structure of the set containing DBCs shown as Lemma 1 where $\Phi(b,t)$ and $\bar{\Phi}(b,t)$ represent $\Phi(b,t,n_{b,t})$ and $\bar{\Phi}(b,t,\bar{n}_{b,t})$ respectively.

Lemma 1 *Let n be a positive integer, $b \geq 0$, $t \geq 0$, and $b+t > 0$. The structure of $\Phi(b,t)$ and that of $\bar{\Phi}(b,t)$ are described as follows.*

1. *If $n_{b,t} < 2^b 3^{t-1}$, i.e., $n_{b,t} = n_{b-1,t} = n_{b,t-1}$, then*

$$\begin{aligned}\Phi(b,t) =& \Phi(b-1,t) \bigcup \left(2^{b-1}3^t \bar{\Phi}(b-1,t)\right) \bigcup \Phi(b,t-1) \bigcup \left(2^b 3^{t-1} \bar{\Phi}(b,t-1)\right),\\ \bar{\Phi}(b,t) =& \left(-2^{b-1}3^t \bar{\Phi}(b-1,t)\right).\end{aligned}$$

2. *If $2^b 3^{t-1} \leq n_{b,t} < 2^{b-1}3^t$, i.e., $n_{b,t} = n_{b-1,t} = n_{b,t-1} + 2^b 3^{t-1}$, then*

$$\begin{aligned}\Phi(b,t) =& \Phi(b-1,t) \bigcup \left(2^{b-1}3^t \bar{\Phi}(b-1,t)\right) \bigcup \left(2^b 3^{t-1} \Phi(b,t-1)\right),\\ \bar{\Phi}(b,t) =& \left(-2^{b-1}3^t \bar{\Phi}(b-1,t)\right) \bigcup \left(-2^b 3^{t-1} \bar{\Phi}(b,t-1)\right).\end{aligned}$$

3. *If $2^{b-1}3^t \leq n_{b,t} < 2 \cdot 2^b 3^{t-1}$, i.e., $n_{b,t} = n_{b-1,t} + 2^{b-1}3^t = n_{b,t-1} + 2^b 3^{t-1}$, then*

$$\begin{aligned}\Phi(b,t) =& \left(2^{b-1}3^t \Phi(b-1,t)\right) \bigcup \left(2^b 3^{t-1} \Phi(b,t-1)\right),\\ \bar{\Phi}(b,t) =& \left(-2^{b-1}3^t \Phi(b-1,t)\right) \bigcup \bar{\Phi}(b-1,t) \bigcup \left(-2^b 3^{t-1} \bar{\Phi}(b,t-1)\right).\end{aligned}$$

4. *If $n_{b,t} \geq 2 \cdot 2^b 3^{t-1}$, i.e., $n_{b,t} = n_{b-1,t} + 2^{b-1}3^t = n_{b,t-1} + 2 \times 2^b 3^{t-1}$, then*

$$\begin{aligned}\Phi(b,t) =& \left(2^{b-1}3^t \Phi(b-1,t)\right),\\ \bar{\Phi}(b,t,) =& \left(-2^{b-1}3^t \Phi(b-1,t)\right) \bigcup \bar{\Phi}(b-1,t) \bigcup \left(-2^b 3^{t-1} \Phi(b,t-1)\right) \bigcup \bar{\Phi}(b,t-1).\end{aligned}$$

The proofs, examples, and remarks can be found in the full version of this paper [26].

The definitions of $n_{b,t}$ and $\bar{n}_{b,t}$ indicate that both $n_{b,t} = n_{b-1,t} = n_{b,t-1} + 2^{b+1}3^{t-1}$ and $n_{b,t} = n_{b-1,t} + 2^{b-1}3^t = n_{b,t-1}$ are impossible. From Lemma 1, $\Phi(b,t)$ and $\bar{\Phi}(b,t)$ only rely on $\Phi(b-1,t)$, $\bar{\Phi}(b-1,t)$, $\Phi(b,t-1)$ and $\bar{\Phi}(b,t-1)$. By the definitions of $n_{b,t}$ and $\bar{n}_{b,t}$, the structure of $\Phi(b,t)$ and that of $\bar{\Phi}(b,t)$ still work for $n_{b,t} = 0$ in Case 1, $n_{b,t} = 2^b 3^{t-1}$ in Case 2, $n_{b,t} = 2^{b-1}3^t$ in Case 3, and $n_{b,t} = 2 \cdot 2^b 3^{t-1}$ in Case 4.

This is the first structure of the set containing all DBCs with a leading term strictly dividing $2^b 3^t$ in the literature. Based on this structure, we will show the number of DBCs with a leading term dividing $2^b 3^t$ for a positive integer n.

3.2 The Number of DBCs

Let $|\mathscr{S}|$ be the cardinality of the set $\mathscr{S}$. The number of DBCs with a leading term dividing $2^b 3^t$ for representing $n_{b,t}$ is $|\Phi(b,t)| + |\bar{\Phi}(b,t)|$. We will provide some initial values of $|\Phi|$ and $|\bar{\Phi}|$. If $n < 0$ or $n \geq 2^b 3^t$ or $b < 0$ or $t < 0$, $|\Phi(b,t,n)| = |\bar{\Phi}(b,t,-n)| = 0$. $|\Phi(0,0,0)| = |\bar{\Phi}(0,0,0)| = 1$.

Based on Lemma 1, the cardinality of $\Phi(b,t)$ and that of $\bar{\Phi}(b,t)$ are shown as Theorem 1.

Theorem 1 *Let n be a positive integer, $b \geq 0$, $t \geq 0$, and $b + t > 0$. We have*

1. *If $n_{b,t} < 2^{b-1}3^{t-1}$, then*

$$\begin{aligned}|\Phi(b,t)| =& |\Phi(b-1,t)| + |\bar{\Phi}(b-1,t)| + |\Phi(b,t-1)| + |\bar{\Phi}(b,t-1)| \\ & - |\Phi(b-1,t-1)| - |\bar{\Phi}(b-1,t-1)|, \\ |\bar{\Phi}(b,t)| =& |\bar{\Phi}(b-1,t)|.\end{aligned}$$

2. *If $2^{b-1}3^{t-1} \leq n_{b,t} < 2^b 3^{t-1}$, then*

$$\begin{aligned}|\Phi(b,t)| =& |\Phi(b-1,t)| + |\bar{\Phi}(b-1,t)| + |\Phi(b,t-1)| \\ & + |\bar{\Phi}(b,t-1)| - |\Phi(b-1,t-1)|, \\ |\bar{\Phi}(b,t)| =& |\bar{\Phi}(b-1,t)|.\end{aligned}$$

3. *If $2^b 3^{t-1} \leq n_{b,t} < 2^{b-1}3^t$, then*

$$\begin{aligned}|\Phi(b,t)| =& |\Phi(b-1,t)| + |\bar{\Phi}(b-1,t)| + |\Phi(b,t-1)|, \\ |\bar{\Phi}(b,t)| =& |\bar{\Phi}(b-1,t)| + |\bar{\Phi}(b,t-1)|.\end{aligned}$$

4. *If $2^{b-1}3^t \leq n_{b,t} < 2 \cdot 2^b 3^{t-1}$, then*

$$\begin{aligned}|\Phi(b,t)| =& |\Phi(b-1,t)| + |\Phi(b,t-1)|, \\ |\bar{\Phi}(b,t)| =& |\Phi(b-1,t)| + |\bar{\Phi}(b-1,t)| + |\bar{\Phi}(b,t-1)|.\end{aligned}$$

5. *If $2 \cdot 2^b 3^{t-1} \leq n_{b,t} < 5 \cdot 2^{b-1}3^{t-1}$, then*

$$\begin{aligned}|\Phi(b,t)| =& |\Phi(b-1,t)|, \\ |\bar{\Phi}(b,t)| =& |\Phi(b-1,t)| + |\bar{\Phi}(b-1,t)| + |\Phi(b,t-1)| \\ & + |\bar{\Phi}(b,t-1)| - |\bar{\Phi}(b-1,t-1)|.\end{aligned}$$

6. *If* $n_{b,t} \geq 5 \cdot 2^{b-1}3^{t-1}$, *then*

$$\begin{aligned}|\Phi(b,t)| =& |\Phi(b-1,t)|,\\ |\bar{\Phi}(b,t)| =& |\Phi(b-1,t)| + |\bar{\Phi}(b-1,t) + |\Phi(b,t-1)|\\ &+ |\bar{\Phi}(b,t-1)| - |\bar{\Phi}(b-1,t-1)| - |\Phi(b-1,t-1)|.\end{aligned}$$

Based on Theorem 1, we have

Corollary 1 *1. If* $b \geq 0$ *and* $t \geq 0$, *then* $|\Phi(b,t)| \geq |\Phi(b-1,t)|$, $|\Phi(b,t)| \geq |\Phi(b,t-1)|$, $|\bar{\Phi}(b,t)| \geq |\bar{\Phi}(b-1,t)|$, *and* $|\bar{\Phi}(b,t)| \geq |\bar{\Phi}(b,t-1)|$.

2. If $b \geq 0$ *and* $t \geq 0$, *then* $|\Phi(b,t)| \leq 4^{b+t}$ *and* $|\bar{\Phi}(b,t)| \leq 4^{b+t}$.

By Corollary 1, $|\Phi(b,t)|$ and $|\bar{\Phi}(b,t)|$ are both $\mathcal{O}(\log n)$-bit integers when both b and t are $\mathcal{O}(\log n)$.

Based on Theorem 1, we employ an iterative algorithm to compute the number of DBCs with a leading term strictly dividing 2^b3^t for $n_{b,t}$ and $\bar{n}_{b,t}$ shown as Algorithm 1. The number of DBCs with a leading term dividing 2^b3^t for n is

1. $|\Phi(b,t)| + |\bar{\Phi}(b,t)|$ when $2^b3^t > n$;
2. $|\Phi(b,t)|$ when $\frac{n}{2} < 2^b3^t \leq n$;
3. 0 when $2^b3^t \leq \frac{n}{2}$.

Algorithm 1 Iterative algorithm to compute the number of DBCs

Input: A positive integer n, $b \geq 0$, and $t \geq 0$
Output: The number of DBCs with a leading term strictly dividing 2^b3^t for $n_{b,t}$ and $\bar{n}_{b,t}$

```
1.   |Φ(0,0)| ← 1, |Φ̄(0,0)| ← 0
2.   For i from 0 to b, |Φ(i,−1)| = |Φ̄(i,−1)| ← 0
3.   For j from 0 to t, |Φ(−1,j)| = |Φ̄(−1,j)| ← 0
4.   For j from 0 to t
5.       For i from 0 to b
6.           If i + j > 0, using Theorem 1 to compute |Φ(i,j)| and |Φ̄(i,j)|
7.   return |Φ(b,t)|, |Φ̄(b,t)|
```

Algorithm 1 terminates in $\mathcal{O}\left((\log n)^3\right)$ bit operations and $\mathcal{O}\left((\log n)^2\right)$ bits of memory when b and t are both in $\mathcal{O}(\log n)$.

Miracl lib [27] is used to implement big number arithmetic. Our experiments in this paper are compiled and executed on Intel® Core™ i7−6567U 3.3 GHZ with Skylake architecture (our algorithms may have different running time on other architectures). Algorithm 1 requires 34, 177, 551, and 1184 million cpu cycles (10, 50, 170, and 360 milliseconds) for 256−bit, 512−bit, 768−bit, and 1024−bit integers respectively. The details are shown in Table 2.

By Algorithm 1, the number of DBCs of $\lfloor \pi \times 10^{120} \rfloor$ with a leading term dividing $2^{240}3^{120}$ is 405694512689803328570475272448020332384436179545046727328115784

Table 2. Cost of Algorithm 1

bits of n	256	512	768	1024
b,t	128,81	256,161	384,242	512,323
cost(million cpu cycles)	34	177	551	1184

367271984621308621154227072670259226179703610530387857487 9. The number of DBCs with a leading term dividing 2^b3^t for 100 when $b < 50$ and $t < 50$ is shown as Table 3. There exist 405 DBCs with a leading term dividing $2^7 3^4$ for representing 100. These results all show a redundance of DBCs for a positive integer. The number of DBCs with a leading term dividing 2^b3^t of 100 is the same for $4 \le t < 50$. For the same b, we guess the number is the same when $t \ge 50$. For each $8 \le b < 50$, the number of DBCs with a leading term dividing 2^b3^t of 100 minus the number of DBCs with a leading term dividing $2^{b-1}3^t$ of 100 is 7. We guess this result is still true for $b \ge 50$.

Table 3. Number of DBCs with a leading term dividing 2^b3^t for 100

	$t=0$	$t=1$	$t=2$	$t=3$	$t<50$
$b=0$	0	0	0	0	1
$b=1$	0	0	0	0	7
$b=2$	0	0	0	11	24
$b=3$	0	0	18	51	70
$b=4$	0	0	57	112	137
$b=5$	0	13	111	188	219
$b=6$	3	35	174	273	310
$b=7$	10	61	241	362	405
$b<50$	$10+7*(b-7)$	$61+26*(b-7)$	$241+67*(b-7)$	$362+89*(b-7)$	$405+95*(b-7)$

3.3 The Number of DBCs for Large b or t

If b or t is large, the number of DBCs are shown as Corollary 2.

Corollary 2 *Let n be a given positive integer, t_τ be a positive integer satisfying $3^{t_\tau-1} > n$ and $3^{t_\tau-2} \le n$, and b_τ be a positive integer satisfying $2^{b_\tau} > 3n$ and $2^{b_\tau-1} \le 3n$. Then*

1. *If $t \ge t_\tau$ and $b \in \mathbb{Z}$, then $|\Phi(b,t)| = |\Phi(b,t_\tau)|$.*
2. *If $b \ge b_\tau$ and $t \in \mathbb{Z}$, then $|\Phi(b,t)| = |\Phi(b_\tau,t)| + (b-b_\tau)C_\tau$ where $C_\tau = \sum_{i=0}^{t} |\bar{\Phi}(b_\tau,i)|$.*
3. *If $b \ge b_\tau$ and $t \ge t_\tau$, then $|\Phi(b,t)| = |\Phi(b_\tau,t)| + (b-b_\tau)C_\tau$ where $C_\tau = \sum_{i=0}^{t_\tau} |\bar{\Phi}(b_\tau,i)|$.*

These three properties of Corollary 2 are used to compute the number of DBCs with a leading term dividing 2^b3^t for some large b and t. The number of DBCs with a leading term dividing 2^b3^t is a constant when $t > t_\tau$. The number of DBCs with a leading term dividing 2^b3^t adds a constant $\sum_{i=0}^{t} |\bar{\Phi}(b_\tau,i)|$ is the number of DBCs with a leading term dividing $2^{b+1}3^t$ when $b > b_\tau$. Take 100 for example, 100 has 137 DBCs with a leading term dividing $2^4 3^t$ for each $t \ge t_\tau$, and has $405+95*(b-7)$ DBCs with a leading term dividing 2^b3^t for each $b \ge 9$ and $t \ge 6$. These results may be associated with that $1 = 2^b - \sum_{i=0}^{b-1} 2^i$ as b becomes larger and that $1 = 3^0$ can not be represented as other ternary representation with its coefficients in $\{\pm 1\}$.

4 Hamming Weight of DBCs

For a positive integer n, Chalermsook, Imai, and Suppakitpaisarn [28] showed that the Hamming weight of unsigned DBNS representations obtained from the greedy approach proposed by Dimitrov, Imbert, and Mishra [1] is $\theta\left(\frac{\log n}{\log\log n}\right)$. And they showed that the Hamming weight of unsigned DBCs produced by greedy approach [1] is $\theta\left(\log n\right)$.

For the Hamming weights of (signed) DBNS representations and DBCs, Dimitrov, Imbert, and Mishra [1] showed that every integer n has a DBNS representation with Hamming weight $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$. Every integer n has a DBC with Hamming weight $\mathcal{O}(\log n)$. These are upper bounds on the Hamming weight of DBNS representations and DBCs. The number of DBCs of a positive integer is infinite and the leading term of its DBC may be infinite. The range of the leading term of canonic DBCs is useful to show the lower bounds of the Hamming weight of DBCs.

4.1 The Range of the Leading Term of Optimal DBCs and Canonic DBCs

Doche [16] proved that a DBC with leading term 2^b3^t belongs to the interval $\left[\frac{3^t+1}{2}, 2^{b+1}3^t - \frac{3^t+1}{2}\right]$. His result showed the range of integers for a leading term. The leading term of a DBC $2^{b_l}3^{t_l}$ for a positive integer does not have an upper bound for $1 = 2^{b_l} - 2^{b_l-1} - \ldots - 2 - 1$ where b_l is an arbitrary positive integer. We will show the range of the leading term of optimal DBCs and that of canonic DBCs for a given integer in Lemma 2.

Lemma 2 *Let n be a positive integer represented as* $\mathrm{w} : \sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i}$, $c_l = 1, c_i \in \{\pm 1\}$ *for* $1 \le i \le l-1$. *Then* $\frac{n}{2} < 2^{b_l}3^{t_l} < 2n$ *when* w *is an optimal DBC, and* $\frac{16n}{21} < 2^{b_l}3^{t_l} < \frac{9n}{7}$ *when* w *is a canonic DBC.*

The range of the leading term of optimal DBCs is useful to prove that the DBC produced by Capuñay and Thériault's algorithm [7] and that produced by Bernstein, Chuengsatiansup, and Lange's algorithm [17] both are optimal DBCs. The leading term of canonic DBCs of n is in the interval $\left(\frac{16n}{21}, \frac{9n}{7}\right)$. It is useful to prove that the DBCs generated by Doche's algorithm is a canonic DBC [16], and to prove the asymptotic lower bound on the Hamming weights of DBCs in the following.

4.2 A Lower Bound on the Hamming Weights of DBCs

Dimitrov and Howe proved that there exist infinitely many integers n whose shortest DBNS representations have Hamming weights $\Omega\left(\frac{\log n}{\log\log n \log\log\log n}\right)$ [29]. The minimum Hamming weight of DBCs for a positive integer n is also called Kolmogorov complexity [30] of a DBC of n, i.e., the Hamming weight of canonic DBCs of n. Lou, Sun, and Tartary [5] proved a similar result for DBCs: there exists at least one $\lfloor \log n \rfloor$ −bit integer such that any DBC representing this integer needs at least $\Omega\left(\lfloor \log n \rfloor\right)$ terms. We will give a stronger result in Lemma 3.

Lemma 3 *For arbitrary* $\alpha \in (0,1)$ *and* $0 < C < \frac{\alpha^2}{8.25}$, *more than* $n - n^\alpha$ *integers in* $[1,n]$ *satisfy that the Hamming weight of the canonic DBCs of each integer is greater than* $C\log n$ *when* $n > N$ *(N is some constant shown as Claim 1).*

For convenience, we first give some conventions and definitions. $s(m)$ denotes the Hamming weight of canonic DBCs of m, and e is the base of the natural logarithm. Let φ_l be the number of DBCs $\sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i}$ with $2^{b_l}3^{t_l} < \frac{9n}{7}, c_i \in \{\pm 1\}$, and $c_l = 1$.

Definition 4 ($\varphi(L)$) *For a given positive integer* n *and a constant* L, $\varphi(L) = \sum_{l=1}^{L} \varphi_l$, *i.e.,* $\varphi(L)$ *is the number of DBCs* $\sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i}$ *with* $2^{b_l}3^{t_l} < \frac{9n}{7}, 1 \le l \le L$.

By Lemma 2, in a canonic DBC, $\frac{16n}{21} < 2^{b_l}3^{t_l} < \frac{9n}{7}$. Then, the number of integers of m in $[1,n]$ represented as a canonic DBC with Hamming weight no greater than L is not more than the number of integers of m in $[1,n]$ represented as a DBC with a leading term dividing $2^{b_l}3^{t_l} < \frac{9n}{7}$, $l \le L$. Since every DBC corresponds to only one integer and each integer has at least one DBC, the number of integers in $[1,n]$ represented as a canonic DBC with Hamming weight no greater than L is no greater than $\varphi(L)$.

An outline of the proof of Lemma 3 is as follows. The number of integers of m in $[1,n]$ can not be represented as a DBC of Hamming weight j, $0 < j \le L$ is equal to n minus the number of integers of m in $[1,n]$ represented in that way. There are at least $n - \varphi(L)$ integers of m in $[1,n]$ can not be represented as a DBC of Hamming weight j with $2^{b_j}3^{t_j} \le \frac{9n}{7}$, $0 < j \le L$. Thus there are at least $n - \varphi(L)$ integers of m in $[1,n]$ satisfying $s(m) > L$. Hence, $\varphi(C\log n) < n^\alpha$ is enough to prove Lemma 3.

Since φ_j where $0 < j \le C\log n$ is the number of DBCs of Hamming weight j with $2^{b_l}3^{t_l} < \frac{9n}{7}$, we have

$$\varphi_j \le 2^{j-1} \sum_{\alpha+\gamma\log 3<\log\frac{9n}{7}} \binom{\alpha+j}{j-1}\binom{\gamma+j}{j-1}.$$

Then

$$\varphi(C\log n) = \sum_{j=1}^{C\log n} \varphi_j \le \sum_{j=1}^{C\log n} \left(2^{j-1} \sum_{\alpha+\gamma\log 3<\log\frac{9n}{7}} \binom{\alpha+j}{j-1}\binom{\gamma+j}{j-1} \right). \tag{1}$$

For this estimate of $\varphi(C\log n)$ is too complex to be dealt with, we simplify its estimate by Claim 1 and its proof requires the tools of Pascal's triangle and Stirling's formula.

Claim 1 *For any* $0 < C < 1$, *when* $n > N$ *where* N *satisfies that* $N > 2^{10000\cdot(3-0.5\log_3 7)}$ *and* $\log N < 1.0001^{C\log N}$,

$$\sum_{j=1}^{C\log n} \left(2^{j-1} \sum_{\alpha+\gamma\log_2 3<\log\frac{9n}{7}} \binom{\alpha+j}{j-1}\binom{\gamma+j}{j-1} \right) < n^{C\log\left(\frac{2.0002e^2(0.5001\log_3 2+C)^2}{C^2}\right)}.$$

According to Equation (1) and Claim 1, we have

$$\varphi(C\log n) < n^{C\log\left(\frac{2.0002e^2\log 3\cdot(0.5001\log_3 2+C)^2}{C^2}\right)}.$$

For some larger N, the coefficients of $\log_3 2$ and e^2 will be smaller than 0.50001 and 2.0002 respectively in this inequation, and for some smaller N, the coefficients of $\log_3 2$ and e^2 will be larger than 0.50001 and 2.0002. The proof of Lemma 3 is as follows.

Proof. To prove Lemma 3, it is sufficient to show that the number of integers of m in $[1,n]$, represented as a DBC of Hamming weight j with $j \le C\log n$ and $2^{b_j}3^{t_j} < \frac{9n}{7}$, is no greater than n^α.

The number of integers of m in $[1,n]$ can be represented as DBCs of Hamming weight j with $2^{b_j}3^{t_j} < \frac{9n}{7}$, $0 < j \le C\log n$ is no greater than $\varphi(C\log n)$. This result is sufficient to show that $\varphi(C\log n) < n^\alpha$, i.e., the number of DBCs of Hamming weight j with $j \le C\log n$ is less than n^α.

Since $\varphi(C\log n) < n^{C\log\left(\frac{2.0002e^2\log 3\cdot(0.5001\log_3 2+C)^2}{C^2}\right)}$, then $n^{C\log\left(\frac{2.0002e^2\log 3\cdot(0.5001\log_3 2+C)^2}{C^2}\right)} < n^\alpha$. We have

$$\frac{2.0002e^2\log 3\cdot(0.5001\log_3 2+C)^2}{C^2} < 2^{\frac{\alpha}{C}}.$$

When $0 < C < \frac{\alpha^2}{8.25}$, this inequality holds.

Thus, for any real numbers α and C with $0 < \alpha < 1$ and $0 < C < \frac{\alpha^2}{8.25}$, when $n > N$, at least $n - n^\alpha$ integers of m in $[1,n]$ satisfy $s(m) > C\log n$.

As a corollary of Lemma 3, for any given positive number $\alpha < 1$, there exist two efficiently computable constants C and N, such that when $n > N$, there are at least $n - n^\alpha$ integers m in $[1,n]$ satisfying $s(m) > C\log n > C\log m$. This result is easy to understand and more advanced than Lou, Sun, and Tartary's result [5].

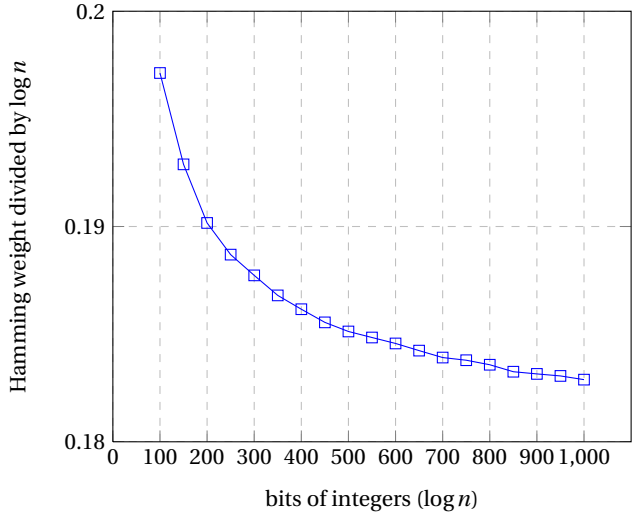
Doche and Habsieger [3] showed that the DBC produced by the tree approach is shorter than that produced by greedy approach experimentally. The average Hamming weight of the DBCs produced by the tree approach is $\frac{\log n}{4.6419}$. Then they posed an open question that the average Hamming weight of DBCs generated by the greedy approach may be not $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$. Lemma 3 is sufficient to solve this question.

The average Hamming weight of DBCs of $(\log n)-$bit integers is the average value of the Hamming weights of the DBCs of all $(\log n)-$bit integers where we choose one DBC for each integer. An asymptotic lower bound of the Hamming weights of DBCs is shown in Theorem 2.

Theorem 2 *An asymptotic lower bound of the average Hamming weights of canonic DBCs for* $(\log n)-$*bit integers is* $\frac{\log n}{8.25}$.

All existing algorithms confirm the asymptotic lower bound of Theorem 2. The average Hamming weight of binary representation is $0.5\log n$, that of NAF is $\frac{\log n}{3}$, that of the DBC produced by binary/ternary approach is $0.2284\log n$ [2], and that of the DBC produced by tree approach is $0.2154\log n$ [3]. The Hamming weights of the DBCs produced by these algorithms are still a long way from the lower bound $\frac{\log n}{8.25}$ in Theorem 2.

Fig. 1. The Hamming weight of canonic DBCs of integers



The average Hamming weight of canonic DBCs of integers is shown as Figure 1. The data is gained by Algorithm 3 which will be given in Section 6 for 1000 random integers for each size. It is $0.19713\log n$ for 100−bit integers, $0.190165\log n$ for 200−bit integers, $0.18773\log n$ for 300−bit integers, $0.186158\log n$ for 400−bit integers, $0.185124\log n$ for 500−bit integers, $0.184568\log n$ for 600−bit integers, $0.183913\log n$ for 700−bit integers, $0.183579\log n$ for 800−bit integers, $0.183153\log n$ for 900−bit integers, $0.182887\log n$ for 1000−bit integers, $0.181867\log n$ for 1500−bit integers, $0.181101\log n$ for 2000−bit integers, $0.180495\log n$ for 2500−bit integers, and $0.179822\log n$ for 3000−bit integers. This value of the Hamming weight given for 3000−bit integers still has a distance from the lower bound given in Theorem 2. The Hamming weight divided by $\log n$ is decreased as the integers become larger.

Our method of calculating the asymptotic lower bound of the average Hamming weight of DBCs may be useful to calculate the asymptotic lower bound of the average Hamming weight of extended DBCs [31] where $\mathscr{C} = \{\pm 1, \pm 3, \ldots\}$.

We will propose an efficient algorithm to generate optimal DBCs.

5 Dynamic Programming Algorithm to Produce Optimal DBCs

Several algorithms were designed to produce near optimal DBCs such as greedy approach [1], binary/ternary approach [2], tree approach [3], and mbNAF [13]. Doche [16] generalized Erdös and Loxton's recursive equation of the number of unsigned chain partition [32] and presented an algorithm to produce a canonic DBC. As Doche's algorithm requires exponential time, in 2015, Capuñay and Thériault [7] generalized tree approach and improved Doche's algorithm to produce a canonic DBC or an optimal DBC in polynomial time, explicitly in $\mathscr{O}\left((\log n)^4\right)$ bit operations

and $\mathcal{O}\left((\log n)^3\right)$ bits of memory. This is the first polynomial algorithm to compute an optimal DBC. In 2017, Bernstein, Chuengsatiansup, and Lange [17] presented a DAG algorithm to produce an optimal DBC in $\mathcal{O}\left((\log n)^{2.5}\right)$ bit operations and $\mathcal{O}\left((\log n)^{2.5}\right)$ bits of memory. Bernstein, Chuengsatiansup, and Lange's algorithm was the state-of-the-art.

We will employ dynamic programming [33] to produce an optimal DBC.

5.1 Basics for Dynamic Programming Algorithm

Dynamic programming [33] solves problems by combining the solutions of subproblems. Optimal substructure and overlapping subproblems are two key characteristics that a problem must have for dynamic programming to be a viable solution technique.

Optimal Substructure We will show our problem has optimal substructure, i.e., an optimal solution to a problem contains optimal solutions to subproblems. First, we define sub-chain.

Definition 5 (Sub-chain) *A DBC* $\sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i}$ *is a sub-chain of a DBC* $\sum_{j=1}^{l_0} a_j 2^{d_j} 3^{e_j}$, *if it satisfies both of the following conditions:*

1. $b_l \le d_{l_0}$, $t_l \le e_{l_0}$, *and* $l \le l_0$;
2. *For each* i *satisfies* $1 \le i \le l$, *there exists one* j *satisfying* $c_i = a_j, b_i = d_j$, *and* $t_i = e_j$.

Let $\mathrm{w}(b,t)$ (resp. $\bar{\mathrm{w}}(b,t)$) be one of the DBCs in $\Phi(b,t)$ (resp. $\bar{\Phi}(b,t)$) with the smallest Hamming weight. The optimal substructure of the problem of finding $\mathrm{w}(b,t)$ (resp. $\bar{\mathrm{w}}(b,t)$) is shown in Lemma 4.

Lemma 4 *Let* $\mathrm{w}(b,t)$ *be a minimal chain for* $n_{b,t}$ *in* $\Phi(b,t)$ *and* $\bar{\mathrm{w}}(b,t)$ *be a minimal chain for* $\bar{n}_{b,t}$ *in* $\bar{\Phi}(b,t)$. *If* $\mathrm{w}(b,t)$ *or* $\bar{\mathrm{w}}(b,t)$ *contains a sub-chain* $\mathrm{w}(i,j)$ *for* $n_{i,j}$, *then* $\mathrm{w}(i,j)$ *is minimal for* $n_{i,j}$ *in* $\Phi(i,j)$; *If* $\mathrm{w}(b,t)$ *or* $\bar{\mathrm{w}}(b,t)$ *contains a sub-chain* $\bar{\mathrm{w}}(i,j)$ *for* $\bar{n}_{i,j}$, *then* $\bar{\mathrm{w}}(i,j)$ *is minimal for* $\bar{n}_{i,j}$ *in* $\bar{\Phi}(i,j)$.

Lemma 4 shows that the problem of finding a minimal chain has optimal substructure. We can partition this problem into subproblems. These subproblems may share the same new problems. For example, subproblems for $n_{b,t-1}$ and subproblems for $n_{b-1,t}$ share the same problems for $n_{b-1,t-1}$ and for $\bar{n}_{b-1,t-1}$.

Overlapping Subproblems When a recursive algorithm revisits the same problem over and over again rather than always generating new problems, we say that the optimization problem has overlapping subproblems. Dynamic programming algorithms typically take advantage of overlapping subproblems by solving each subproblem once and then storing the solution in a table where it can be looked up when needed.

Based on Lemma 1, using the range of the leading term of a canonic DBC in Lemma 2, we simplify the possible sources of $\mathrm{w}(b,t)$ and $\bar{\mathrm{w}}(b,t)$ shown as Lemma 5.

Lemma 5 *Let n be a positive integer, $b \geq 0$, $t \geq 0$, and $b+t>0$.*

1. *If $\frac{n_{b,t}}{2^{b-1}3^{t-1}} < 2$, then*

$$\begin{aligned}\mathrm{w}(b,t) =& \mathrm{minL}\left\{\mathrm{w}(b-1,t), \mathrm{w}(b,t-1), 2^b 3^{t-1} + \bar{\mathrm{w}}(b,t-1)\right\},\\ \bar{\mathrm{w}}(b,t) =& -2^{b-1}3^t + \bar{\mathrm{w}}(b-1,t).\end{aligned}$$

2. *If $2 \leq \frac{n_{b,t}}{2^{b-1}3^{t-1}} < 3$, then*

$$\begin{aligned}\mathrm{w}(b,t) =& \mathrm{minL}\left\{\mathrm{w}(b-1,t), 2^{b-1}3^t + \bar{\mathrm{w}}(b-1,t), 2^b 3^{t-1} + \mathrm{w}(b,t-1)\right\},\\ \bar{\mathrm{w}}(b,t) =& -2^{b-1}3^t + \bar{\mathrm{w}}(b-1,t).\end{aligned}$$

3. *If $3 \leq \frac{n_{b,t}}{2^{b-1}3^{t-1}} < 4$, then*

$$\begin{aligned}\mathrm{w}(b,t) =& 2^{b-1}3^t + \mathrm{w}(b-1,t),\\ \bar{\mathrm{w}}(b,t) =& \mathrm{minL}\left\{-2^{b-1}3^t + \mathrm{w}(b-1,t), \bar{\mathrm{w}}(b-1,t), -2^b 3^{t-1} + \bar{\mathrm{w}}(b,t-1)\right\}.\end{aligned}$$

4. *If $\frac{n_{b,t}}{2^{b-1}3^{t-1}} \geq 4$, then*

$$\begin{aligned}\mathrm{w}(b,t) =& 2^{b-1}3^t + \mathrm{w}(b-1,t),\\ \bar{\mathrm{w}}(b,t) =& \mathrm{minL}\left\{\bar{\mathrm{w}}(b-1,t), -2^b 3^{t-1} + \mathrm{w}(b,t-1), \bar{\mathrm{w}}(b,t-1)\right\}.\end{aligned}$$

We give some conventions for initial values of $\mathrm{w}(b,t)$ and $\bar{\mathrm{w}}(b,t)$. If $b<0$ or $t<0$, $\mathrm{w}(b,t) = \bar{\mathrm{w}}(b,t) = \mathrm{NULL}$. If $b \geq 0$, $t \geq 0$, and $n_{b,t} = 0$, then $\mathrm{w}(b,t) = \{0\}$ and $\bar{\mathrm{w}}(b,t) = \mathrm{NULL}$.

Lemma 5 reveals the relationship between problems of finding $\mathrm{w}(b,t)$ and $\bar{\mathrm{w}}(b,t)$ and problems of finding their subproblems. Dynamic programming is efficient when a given subproblem may arise from more than one partial set of choices. Each problem of finding $\mathrm{w}(b,t)$ and $\bar{\mathrm{w}}(b,t)$ has at most 4 partial sets of choices. The key technique in the overlapping subproblems is to store the solution of each such subproblem in case it should reappear.

5.2 Dynamic Programming to Compute an Optimal DBC

The main blueprint of our dynamic programming algorithm to produce an optimal DBC contains four steps.

1. Characterize the structure of an optimal solution whose two key ingredients are optimal substructure and overlapping subproblems.
2. Recursively define the value of an optimal solution by minL.

3. Compute a DBC with the smallest Hamming weight and its leading term dividing 2^b3^t for each $n_{b,t}$ and $\bar{n}_{b,t}$ in a bottom-up fashion.
4. Construct an optimal DBC from computed information.

The dynamic programming algorithm to compute an optimal DBC is shown as Algorithm 2. In Algorithm 2, set $B = 2n$ in general cases, and set $B = \frac{9n}{7}$ in the case $D = T = 0$ by Lemma 2.

Algorithm 2 Dynamic programming to compute an optimal DBC

Input: a positive integer n, its binary representation n_{binary}, three non-negative constants $A > 0, D \geq 0, T \geq 0$

Output: an optimal DBC for n

1. **If** $D = 0$ and $T = 0$, $B \leftarrow \frac{9n}{7}$, **else** $B \leftarrow 2n$. $\mathrm{w}(0,0) \leftarrow 0$, $\bar{\mathrm{w}}(0,0) \leftarrow$ NULL, $\mathrm{w}_{\min} \leftarrow n_{\text{binary}}$
2. **For** b **from** 0 **to** $\lfloor \log B \rfloor$, $\mathrm{w}(b,-1) \leftarrow$ NULL, $\bar{\mathrm{w}}(b,-1) \leftarrow$ NULL
3. **For** t **from** 0 **to** $\lfloor \log_3 B \rfloor$, $\mathrm{w}(-1,t) \leftarrow$ NULL, $\bar{\mathrm{w}}(-1,t) \leftarrow$ NULL, $\mathrm{bBound}[t] \leftarrow \left\lfloor \log \frac{B}{3^t} \right\rfloor$
4. **For** t **from** 0 **to** $\lfloor \log_3 B \rfloor$
5. **For** b **from** 0 **to** $\mathrm{bBound}[t]$
6. **If** $b + t > 0$, compute $\mathrm{w}(b,t)$ and $\bar{\mathrm{w}}(b,t)$ using Lemma 5
7. **If** $n > n_{b,t}$, $\mathrm{w}_{\min} \leftarrow \mathrm{minV}\left\{2^b3^t + \mathrm{w}(b,t), \mathrm{w}_{\min}\right\}$
8. **else if** $n = n_{b,t}$, $\mathrm{w}_{\min} \leftarrow \mathrm{minV}\left\{\mathrm{w}(b,t), 2^b3^t + \bar{\mathrm{w}}(b,t), \mathrm{w}_{\min}\right\}$
9. **return** $\mathrm{w}_{\min}$

In Lines $1-3$ of Algorithm 2, the initial values of $\mathrm{w}(0,0)$, $\bar{\mathrm{w}}(0,0)$, $\mathrm{w}_{\min}$, $\mathrm{w}(b,-1)$, $\bar{\mathrm{w}}(b,-1)$, $\mathrm{w}(-1,t)$ and $\bar{\mathrm{w}}(-1,t)$ are given. $\mathrm{w}_{\min}$ stores the resulting DBC for n whose initial value is n_{binary}, i.e., the binary representation of n.

In the Lines $4-8$ of Algorithm 2, a two-layer cycle computes a DBC $\mathrm{w}_{\min}$. Line 6 shows that the problem of computing $\mathrm{w}(b,t)$ and $\bar{\mathrm{w}}(b,t)$ are partitioned into subproblems of computing $\mathrm{w}(b-1,t)$, $\bar{\mathrm{w}}(b-1,t)$, $\mathrm{w}(b,t-1)$, and $\bar{\mathrm{w}}(b,t-1)$ using Lemma 5. This is a bottom-up fashion. For the same t, we compute $\mathrm{w}(0,t)$ (the same for $\bar{\mathrm{w}}(0,t)$); next, compute $\mathrm{w}(1,t)$, …, $\mathrm{w}\left(\left\lfloor \log \frac{B}{3^t} \right\rfloor, t\right)$. Since $\mathrm{w}(b,t-1)$ and $\bar{\mathrm{w}}(b,t-1)$ have been computed by Lines 4 and 6 in the last loop of t and $\mathrm{w}(b-1,t)$ and $\bar{\mathrm{w}}(b-1,t)$ have been computed by Lines 5 and 6 in the last loop of b, we compute $\mathrm{w}(b,t)$ and $\bar{\mathrm{w}}(b,t)$ successfully. Using these results to solve the subproblems recursively, we can avoid calculating a problem twice or more.

By Lemma 4 and the bottom-up fashion, $\mathrm{w}(b,t)$ and $\bar{\mathrm{w}}(b,t)$ have been computed by Algorithm 2 for all b and t satisfying $2^b3^t < B$. We will show that the DBC returned by Algorithm 2 is an optimal DBC in Theorem 3.

Theorem 3 *Algorithm 2 produces a canonic DBC when $D = T = 0$, and an optimal DBC when $D + T > 0$.*

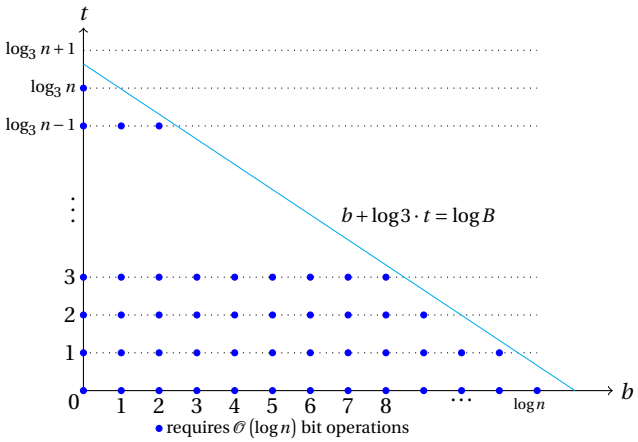
If one wants to generate a different optimal DBC or canonic DBC, one possibility is to adjust the function minL and minV when two or more DBCs have the same value. Doing this, we can favor doubling or tripling. In our algorithm, we favor tripling.

Optimal DBCs are usually varied with Hamming weight by different costs of point operations. Canonic DBCs returned by Algorithm 2 are with the same Hamming weight and are not affected by the cost of point operations. Take a positive integer $\lfloor \pi \times 10^{20} \rfloor = 314159265358979323846$ for example. Its optimal DBC returned by Algorithm 2 is $2^{30}3^3+2^{28}3^2+2^{20}3^2-2^{17}3^1-2^{16}3^0-2^8 3^0+2^3 3^0-2^0 3^0$ with Hamming weight 8 for $\mathscr{E}_{\mathrm{W}}$ 0.8. The value of the cost of this DBC is 319.2. Its optimal DBC returned by Algorithm 2 is $2^{19}3^{10}+2^{13}3^{10}-2^{12}3^8+2^9 3^6+2^6 3^5+2^3 3^2-2^0 3^0$ with Hamming weight 7 for $\mathscr{E}_{\mathrm{W}}$ 1. The value of the cost of this DBC is 358. This DBC with Hamming weight 7 is one of the canonic DBCs of $\lfloor \pi \times 10^{20} \rfloor$.

5.3 The Time Complexity and Space Complexity of Algorithm 2

The running time of a dynamic programming algorithm depends on the product of two factors: the number of subproblems overall and how many choices we look at for each subproblem. Our dynamic programming algorithm has $(\log n+1)(\log_3 n+1)$ subproblems. If we store the value of $n_{b,t}$ and $n/(2^b 3^t)$ for the use of next cycle, each subproblems requires $\mathscr{O}(\log n)$ bit operations. Algorithm 2 terminates in $\mathscr{O}\left((\log n)^3\right)$ bit operations. The details are illustrated by Figure 2. Each node (b,t) of computing $\left\lfloor \frac{n_{b,t}}{2^{b-1}3^{t-1}} \right\rfloor$, $\mathrm{w}(b,t)$, and $\bar{\mathrm{w}}(b,t)$ requires $\mathscr{O}(\log n)$ bit operations.

Fig. 2. The procedure of our dynamic programming algorithm



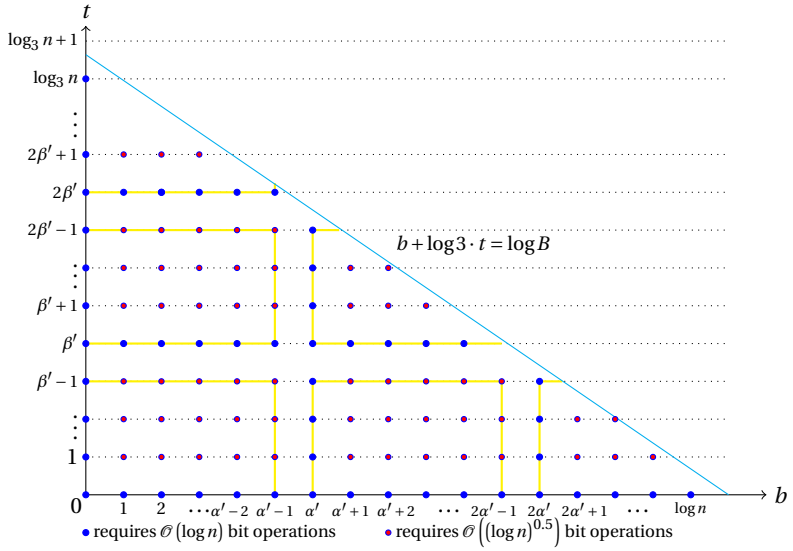
If the powers of 2 and 3 are recorded by their differences as Remark 5 of Capuñay and Thériault's work [6], our algorithm terminates in $\mathscr{O}\left((\log n)^2\right)$ bits of memory. The details are shown as follows. The term $c_i 2^{b_i} 3^{t_i}$ in the chain is stored as the pair (c_i, b_i, t_i). For example, $1000 = 2^{10}-2^5+2^3$ is recorded as $(1,3,0)$, $(-1,2,0)$, and $(1,5,0)$. If DBCs are recorded as their difference with the previous term, then the memory requirement per chain is $\mathscr{O}(\log n)$. Thus, Algorithm 2 requires $\mathscr{O}\left((\log n)^2\right)$ bits of memory.

We will focus on improving the time complexity of Algorithm 2.

6 Equivalent Representatives for Large Numbers

The most time-consuming part of Lemma 5 is to compute $\frac{n_{b,t}}{2^{b-1}3^{t-1}}$. It can be improved by reduced representatives for large numbers [17]. Bernstein, Chuengsatiansup, and Lange [17] noticed that arbitrary divisions of $\mathcal{O}(\log n)$−bit numbers take time $(\log n)^{1+o(1)}$ shown in pages 81−86 of "on the minimum computation time of functions" by Cook [34]. Based on this novel representative, the time complexity of dynamic programming algorithm is shown as Figure 3. In Figure 3, $\alpha' = (\log B)^{0.5}$ and $\beta' = (\log_3 B)^{0.5}$. Each node (b,t) satisfying $\alpha' | b$ or $\beta' | t$ is named a boundary node in Figure 3. Each boundary node requires $\log n$ bit operations and each of the other nodes requires $(\log n)^{0.5}$ bit operations. Then Algorithm 2 terminates in $\mathcal{O}\left((\log n)^{2.5}\right)$ bit operations using reduced representatives.

Fig. 3. The procedure of our dynamic programming algorithm using the trick in [17]



Motivated by their reduced representatives for large numbers, we will give a new representative named equivalent representative.

Definition 6 (Equivalent representative) *If one expression of an integer n' is equal to the value of $\left\lfloor \frac{n_{b,t}}{2^{b-1}3^{t-1}} \right\rfloor$ in Lemma 5, then n' is an equivalent representative of n.*

Our equivalent representative is a generalization of Bernstein, Chuengsatiansup, and Lange's reduced representative. Reduced representatives for large numbers do not work for $\log n + \log_3 n$ boundary nodes. Our equivalent representatives will solve this problem.

6.1 Use Equivalent Representatives in Algorithm 2

We employ equivalent representatives to improve the recode procedure of Algorithm 2 shown as Algorithm 3. n_1 is an equivalent representative in Algorithm 3 shown by Claim 2.

Claim 2 *Let* $n_1{}' = \left\lfloor \frac{6\cdot n}{2^{\mathrm{ii}_1\cdot\alpha_1^2}3^{\mathrm{jj}_1\cdot\beta_1^2}} \right\rfloor \% \left(2^{\alpha_1^2+1}3^{\beta_1^2+1}\right)$, $n_1 = \left\lfloor \frac{n_1{}'}{2^{i_1\cdot\alpha_1}3^{j_1\cdot\beta_1}} \right\rfloor \% \left(2^{\alpha_1+1}3^{\beta_1+1}\right)$, $\alpha_1 = \left\lfloor (\log B)^{\frac{1}{3}} \right\rfloor$, $\beta_1 = \left\lfloor (\log B)^{\frac{1}{3}} \right\rfloor$, $b = \mathrm{ii}_1\cdot\alpha_1^2 + i_1\cdot\alpha_1 + i$, $t = \mathrm{jj}_1\cdot\beta_1^2 + j_1\cdot\beta_1 + j$, $i_1 \geq 0$, $j_1 \geq 0$, $0 \leq i < \alpha$, $0 \leq j < \beta$ *shown as Algorithm 3. Then* $\left(\left\lfloor \frac{n_1}{2^i3^j} \right\rfloor \%6\right) = \left\lfloor \frac{n_{b,t}}{2^{b-1}3^{t-1}} \right\rfloor$.

Algorithm 3 Dynamic programming to compute an optimal DBC using equivalent representatives once

---

Input: a positive integer n and its binary representation n_{binary}, three non-negative constants $A > 0, D \geq 0, T \geq 0$
Output: an optimal DBC for n

1. Lines 1 – 3 of Algorithm 2
2. $\alpha_0 \leftarrow \lfloor \log B \rfloor$, $\beta_0 \leftarrow \lfloor \log_3 B \rfloor$, $\alpha_1 \leftarrow \left\lfloor (\log B)^{\frac{1}{3}} \right\rfloor$, $\beta_1 \leftarrow \left\lfloor (\log B)^{\frac{1}{3}} \right\rfloor$
3. **For** jj_1 **from** 0 **to** $\left\lfloor \frac{\log_3 B}{\beta_1^2} \right\rfloor + 1$
4. **For** ii_1 **from** 0 **to** $\left\lfloor \frac{\mathrm{bBound}[j\cdot\beta_1^2]}{\alpha_1^2} \right\rfloor + 1$
5. $n_1{}' \leftarrow \left\lfloor \frac{6\cdot n}{2^{\mathrm{ii}_1\cdot\alpha_1^2}3^{\mathrm{jj}_1\cdot\beta_1^2}} \right\rfloor \% \left(2^{\alpha_1^2+1}3^{\beta_1^2+1}\right)$
6. **For** j_1 **from** 0 **to** $\beta_1 - 1$
7. **For** i_1 **from** 0 **to** $\alpha_1 - 1$
8. $n_1 \leftarrow \left\lfloor \frac{n_1{}'}{2^{i_1\cdot\alpha_1}3^{j_1\cdot\beta_1}} \right\rfloor \% \left(2^{\alpha_1+1}3^{\beta_1+1}\right)$
9. **For** j **from** 0 **to** $\beta_1 - 1$
10. **For** i **from** 0 **to** $\alpha_1 - 1$
11. $t \leftarrow \mathrm{jj}_1\cdot\beta_1^2 + j_1\cdot\beta_1 + j, b \leftarrow \mathrm{ii}_1\cdot\alpha_1^2 + i_1\cdot\alpha_1 + i$
12. **If** $b + t > 0$ & $b <$ bBound[t] & $t \leq \lfloor \log_3 B \rfloor$
13. compute $\mathrm{w}(b,t)$, $\bar{\mathrm{w}}(b,t)$ using Lemma 5 ▷ $\left\lfloor \frac{n_{b,t}}{2^{b-1}3^{t-1}} \right\rfloor$ is calculated by $\left(\left\lfloor \frac{n_1}{2^i3^j} \right\rfloor \%6\right)$
14. **else if** $b =$ bBound[t] & $t \leq \lfloor \log_3 B \rfloor$, Lines 7, 8 of Algorithm 2
15. **return** $\mathrm{w}_{\min}$

Notice that $t = \mathrm{jj}_1\cdot\beta_1^2 + j_1\cdot\beta_1 + j$, $b = \mathrm{ii}_1\cdot\alpha_1^2 + i_1\cdot\alpha_1 + i$ in Line 11 of Algorithm 3. Algorithm 3 is similar as Algorithm 2 whose total cycles are at most $\log B \log_3 B$.

Algorithm 3 uses a trick of an equivalence representative n_1. The middle variable n_1' is used to calculate the equivalent representative n_1. Each n_1' is a $\mathcal{O}(\alpha_1^2)$-bit integers shown as Algorithm 3. There are at most $\left(\left\lfloor \frac{\log_3 B}{\beta_1^2} \right\rfloor + 1\right)\left(\left\lfloor \frac{\log B}{\alpha_1^2} \right\rfloor + 1\right)$ such

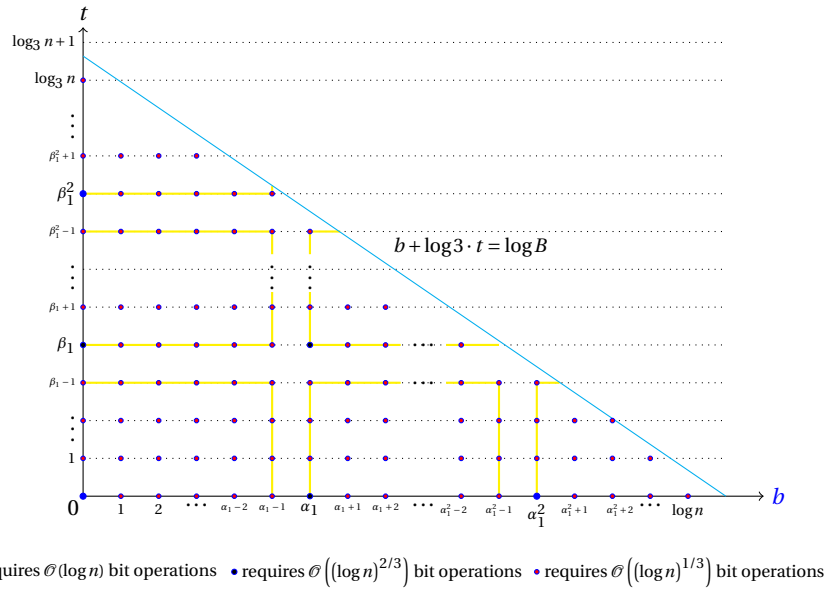
numbers n'_1, i.e., $\mathcal{O}\left(\alpha_1^2\right)$. Calculating each n'_1 requires $\mathcal{O}\left(\log n\right)$ bit operations. Calculating all n'_1 requires $\mathcal{O}\left(\left(\log n\right)^{\frac{5}{3}}\right)$ bit operations. Calculating each representative n_1 requires $\mathcal{O}\left(\alpha_1^2\right)$ bit operations. Then calculating equivalent representatives requires $\mathcal{O}\left(\left(\log n\right)^2\right)$ bit operations.

Based on equivalent representatives, each node (b,t) requires $\mathcal{O}\left(\alpha_1\right)$ bit operations. $\left(\log B\right)\cdot\left(\log_3 B\right)$ nodes requiring $\mathcal{O}\left(\left(\log n\right)^{\frac{7}{3}}\right)$ bit operations. The time complexity of Algorithm 3 is shown in Lemma 6.

Lemma 6 *Algorithm 3 terminates in* $\mathcal{O}\left(\left(\log n\right)^{2+\frac{1}{3}}\right)$ *bit operations.*

The details of the time cost of Algorithm 3 are shown as Figure 4.

Fig. 4. The procedure of Algorithm 3 using equivalent representatives



Based on Algorithm 3, we will use equivalent representatives repeatedly.

6.2 Dynamic Programming using Equivalent Representatives *k*-th

We generate Algorithm 3 and use equivalent representatives k-th in Algorithm 2 shown as Algorithm 4. $\left\lfloor \frac{n_{b,t}}{2^{b-1}3^{t-1}} \right\rfloor$ in Lemma 5 is calculated by $\left(\left\lfloor \frac{n_k}{2^i 3^j} \right\rfloor \%6\right)$. Algorithm 3 is a special case of Algorithm 4 with $k=1$.

The time complexity of Algorithm 4 is shown in Theorem 4.

Theorem 4 *Algorithm 4 terminates in* $\mathcal{O}\left(\left(\log n\right)^2\left(\left(\log n\right)^{\frac{1}{3^k}} + k + \log\log n\right)\right)$ *bit operations. It requires* $\mathcal{O}\left(\left(\log n\right)^2 \log\log n\right)$ *bit operations when* $k = \log_3 \log n$.

Notice that $\alpha_2 \leq 7$ when $n < 2^{134217728}$. Then k in Algorithm 4 is usually 1 or 2. Algorithms 2, 3, and 4 generate the same DBC with the same A, D, T, and n.

Algorithm 4 Dynamic programming to compute an optimal DBC using equivalent representatives k-th

Input: a positive integer n, a positive integer k, and its binary representation n_{binary}, three non-negative constants $A > 0, D \geq 0, T \geq 0$
Output: an optimal DBC for n
1. Lines $1-3$ of Algorithm 2, $n_0 \leftarrow 6 \cdot n$
2. **For** y **from** 0 **to** k, $\alpha_y \leftarrow \left\lfloor (\log B)^{\frac{1}{3^y}} \right\rfloor, \beta_y \leftarrow \left\lfloor (\log_3 B)^{\frac{1}{3^y}} \right\rfloor$
3. **For** jj_y **from** 0 **to** $\left\lfloor \frac{\beta_{y-1}}{\beta_y^2} \right\rfloor + 1$
4. **For** ii_y **from** 0 **to** $\left\lfloor \frac{\alpha_{y-1}}{\alpha_y^2} \right\rfloor + 1$
5. $n_y' \leftarrow \left\lfloor \frac{n_{y-1}}{2^{\text{ii}_y \cdot \alpha_y^2} 3^{\text{jj}_y \cdot \beta_y^2}} \right\rfloor \% \left(2^{\alpha_y^2+1} 3^{\beta_y^2+1}\right)$
6. **For** j_y **from** 0 **to** $\beta_y - 1$
7. **For** i_y **from** 0 **to** $\alpha_y - 1$
8. $n_y \leftarrow \left\lfloor \frac{n_y'}{2^{i_y \cdot \alpha_y} 3^{j_y \cdot \beta_y}} \right\rfloor \% \left(2^{\alpha_y+1} 3^{\beta_y+1}\right)$

▷ **For each** y from 1 to k, Lines 3-8 are repeatedly as y is outer loop and $y+1$ is inner loop

9. **For** j **from** 0 **to** $\beta_k - 1$
10. **For** i **from** 0 **to** $\alpha_k - 1$
11. $t \leftarrow \sum_{y=1}^{k} \left(\text{jj}_y \cdot \beta_y^2 + j_y \cdot \beta_y\right) + j, b \leftarrow \sum_{y=1}^{k} \left(\text{ii}_y \cdot \alpha_y^2 + i_y \cdot \alpha_y\right) + i$
12. **If** $b + t > 0$ & $b <$ bBound$[t]$ & $t \leq \lfloor \log_3 B \rfloor$
13. compute $\text{w}(b,t)$, $\bar{\text{w}}(b,t)$ using Lemma 5
▷ $\left\lfloor \frac{n_{b,t}}{2^{b-1}3^{t-1}} \right\rfloor$ is calculated by $\left(\left\lfloor \frac{n_k}{2^i 3^j} \right\rfloor \%6\right)$
14. **else if** $b =$ bBound$[t]$ & $t \leq \lfloor \log_3 B \rfloor$, Lines 7,8 of Algorithm 2
15. **return** $\text{w}_{\min}$

6.3 Comparison of These Algorithms

The time complexity, space complexity, and method of Doche's algorithm [16], Capuñay and Thériault's algorithm [7], Bernstein , Chuengsatiansup, and Lange's algorithm [17], and Algorithms $2-4$ are summarized in Table 4. Table 4 shows the advantage of our dynamic programming algorithms.

Table 4. Comparison of algorithms to generate optimal DBCs

algorithm	time complexity ($\mathcal{O}$)	space complexity ($\mathcal{O}$)	method
Doche [16]	exponential	$(\log n)^2$	enumeration
CT [7]	$(\log n)^4$	$(\log n)^3$	two cycles
BCL [17]	$(\log n)^{2.5}$	$(\log n)^{2.5}$	DAG
Algorithm 2 (new)	$(\log n)^3$	$(\log n)^2$	dynamic programming
Algorithm 3 (new)	$(\log n)^{2+\frac{1}{3}}$	$(\log n)^2$	using equivalent representatives
Algorithm 4 (new)	$(\log n)^2 \log\log n$	$(\log n)^2$	using equivalent representatives $(\log_3 \log n)$−th

From the time costs of different algorithms to generate optimal DBCs in Table 5, Algorithm 4 is about 20, 25, 28, 32, and 40 times faster than Capuñay and Thériault's

algorithm and 6.1, 6.6, 7.7, 8.7, and 9.3 times faster than Bernstein, Chuengsatiansup and Lange's algorithm for each size ranges in 256, 384, 512, 640, and 768 respectively. As the integer becomes larger, Algorithm 4 will gain more compared to Bernstein, Chuengsatiansup and Lange's algorithm.

Table 5. Time Costs of different algorithms to generate optimal DBCs in million cpu cycles for integers with different size

	256−bit	384−bit	512−bit	640−bit	768−bit
CT [7]	41.9	106	217	386	645
BCL [17]	12.1	28.9	60.1	108	164
Algorithm 4 (new)	1.98	4.32	7.72	11.8	18.0

6.4 The Hamming Weights and Leading Terms of Canonic DBCs and Optimal DBCs

The Hamming weights and leading terms of the DBC produced by greedy approach [1] (greedy-DBC), canonic DBCs, and optimal DBCs are shown in Table 6 for the same 1000 integers by Algorithm 3. The Hamming weight of NAF is $\frac{\log n}{3}$. The Hamming weight of mbNAF, that of the DBC produced by binary/ternary approach(bt-DBC), and that of the DBC produced by tree approach (tree-DBC) are $0.2637\log n$, $0.2284\log n$, and $0.2154\log n$ respectively and the leading terms are $2^{0.791\log n}3^{0.1318\log n}$, $2^{0.4569\log n}3^{0.3427\log n}$, and $2^{0.5569\log n}3^{0.2796\log n}$ respectively. The Hamming weights of canonic DBCs are usually smaller than those of optimal DBCs. By Table 6, the Hamming weights of optimal DBCs are over 60% smaller than those of NAFs. As the integer becomes larger, the Hamming weight dividing $\log n$ will be smaller with a limitation $\frac{1}{8.25}$ by Theorem 2. Please refer to Figure 1 to get more details of the Hamming weight of canonic DBCs.

Table 6. Hamming weights and leading terms of optimal DBCs on elliptic curves with different size

		256−bit	384−bit	512−bit	640−bit	768−bit
greedy-DBC [1]	Hamming weight	62.784	94.175	125.48	155.307	188.764
	leading term(b_l, t_l)	124.282, 82.168	183.256, 125.779	258.908, 159.309	314.954, 204.158	384.604, 240.957
canonic DBC	Hamming weight	48.319	71.572	94.75	118.108	141.097
	leading term(b_l, t_l)	128.275, 80.316	197.183, 117.582	261.227, 157.903	328.541, 196.231	396.162, 234.330
optimal DBC	Hamming weight	50.027	74.163	98.234	122.544	146.493
$\mathscr{E}_W$ 0.8	leading term(b_l, t_l)	176.675, 49.750	265.369, 74.549	353.175, 99.895	444.538, 123.015	532.690, 148.162
optimal DBC	Hamming weight	49.393	73.210	96.993	121.134	144.684
$\mathscr{E}_W$ 1	leading term(b_l, t_l)	169.026, 54.578	253.989, 81.731	338.509, 109.154	426.218, 134.577	509.540, 162.764

We will discuss scalar multiplications using our optimal DBCs.

7 Comparison of Scalar Multiplications

The scalar multiplication algorithm using a DBC is a Horner-like scheme for the evaluation of nP utilizing the DBC of $n = \sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i}$ as $nP = \sum_{i=1}^{l} c_i 2^{b_i} 3^{t_i} P$.

Theoretical cost of scalar multiplications on elliptic curves using NAF, greedy-DBC, bt-DBC, mbNAF, tree-DBC, canonic DBC, and optimal DBC on $\mathscr{E}_{\mathrm{W}}$ 0.8 and $\mathscr{E}_{\mathrm{W}}$ 1 are shown in Table 7.

Table 7 shows that scalar multiplication using an optimal DBC is more efficient than that using a canonic DBC. Scalar multiplication using an optimal DBC on $\mathscr{E}_{\mathrm{W}}$ 0.8 and $\mathscr{E}_{\mathrm{W}}$ 1 is about 13% and 13% faster than that using NAF, 7.5% and 7.1% faster than that using greedy-DBC, 6.5% and 6% faster than that using bt-DBC, 7% and 7% faster than that using mbNAF, 4% and 4% faster than that using a tree-DBC, and 0.9% and 0.7% faster than that using a canonic DBC respectively. Scalar multiplication using an optimal DBC is usually faster than that using a canonic DBC. Take $\lfloor \pi \times 10^{240} \rfloor$ on $\mathscr{E}_{\mathrm{W}}$ 1 for example, scalar multiplication using our optimal DBC is 14% faster and 3.8% faster than that using NAF and tree-DBC respectively.

Table 7. Theoretical costs of scalar multiplications on elliptic curves using optimal DBC, canonic DBC, tree-DBC, and NAF in **M**

bits of n	representation	256−bit	384−bit	512−bit	640−bit	768−bit
$\mathscr{E}_{\mathrm{W}}$ 0.8	NAF	2652	3983	5315	6646	7977
	greedy-DBC [1]	2535	3818	5089	6351	7643
	bt-DBC [2]	2510	3771	5031	6291	7552
	mbNAF [13]	2521	3787	5052	6318	7583
	tree-DBC [3]	2452	3683	4914	6146	7377
	canonic DBC(this work)	2393	3582	4774	5967	7155
	optimal DBC(this work)	2364	3543	4722	5902	7080
$\mathscr{E}_{\mathrm{W}}$ 1	NAF	2976	4469	5962	7456	8949
	greedy-DBC [1]	2824	4252	5671	7075	8516
	bt-DBC [2]	2796	4200	5603	7007	8410
	mbNAF [13]	2824	4241	5659	7076	8494
	tree-DBC [3]	2738	4113	5488	6862	8237
	canonic DBC(this work)	2671	4000	5332	6664	7991
	optimal DBC(this work)	2649	3970	5292	6615	7936

In Table 7, the value of $\frac{T}{D}$ on $\mathscr{E}_{\mathrm{W}}$ 0.8 is greater than that on $\mathscr{E}_{\mathrm{W}}$ 1. The ratio of the cost of scalar multiplication using an optimal DBC to that using NAF on $\mathscr{E}_{\mathrm{W}}$ 0.8 is greater than that on $\mathscr{E}_{\mathrm{W}}$ 1 for integers of each size in Table 7. The ratio of the improvement of scalar multiplication using an optimal DBC compared to NAF is increasing as the value of $\frac{T}{D}$ becomes larger.

A constant-time software implementation is used to protect the scalar multiplication algorithms for avoiding some side-channel attacks by side channel atomicity. Multiplication and squaring are both executed by one multiplication and two additions. For each size ranges in 256, 384, 512, 640, and 768, we generate a prime number p with the same size and create a random curve for $\mathscr{E}_{\mathrm{W}}$ over a finite field $\mathbb{F}_p$. Scalar multiplications using NAF, greedy-DBC, bt-DBC, mbNAF, tree-DBC, canonic DBC, and optimal DBC are shown in Table 8.

Experimental results show that scalar multiplication using an optimal DBC is 13% faster than that using NAF, 7% faster than that using greedy-DBC, 6% faster than that using bt-DBC, 7% faster than that using mbNAF, and 4.1% faster than that using a tree-DBC on $\mathscr{E}_{\mathrm{W}}$ respectively. Within the bounds of the errors, the practical implementations are consistent with these theoretical analyses. The theoretical

Table 8. Experimental cost of scalar multiplications on elliptic curves using optimal DBC, canonic DBC, tree-DBC, and NAF on $\mathscr{E}_{\mathrm{W}}$ in million cpu cycles

representation	256−bit	384−bit	512−bit	640−bit	768−bit
NAF	4.038	8.151	13.94	22.34	34.05
greedy-DBC [1]	3.836	7.751	13.27	21.23	32.43
bt-DBC [2]	3.798	7.656	13.12	21.02	32.03
mbNAF [13]	3.837	7.731	13.25	21.23	32.35
tree-DBC [3]	3.734	7.575	12.92	20.68	31.54
canonic DBC(this work)	3.624	7.279	12.44	19.95	30.35
optimal DBC(this work)	3.594	7.168	12.37	19.83	30.17

analyses and practical implementations both show that the Hamming weight is not the only factor affecting the efficiency of scalar multiplications and that scalar multiplications using optimal DBCs are the fastest.

Those computations do not take the time of producing the expansions into account. The recoding of our optimal DBC takes up a small amount of time to compute scalar multiplication where both take time $\mathscr{O}\left((\log n)^2 \log\log n\right)$ when field multiplications use FFTs. It can't be ignored. Optimal DBCs are suitable for computing scalar multiplications when the multiplier n is fixed.

8 Conclusion

We first proposed a polynomial time algorithm to compute the number of DBCs for a positive integer with a leading term dividing 2^b3^t. We showed theoretical results of the number of DBCs for large b and t and gave an estimate of this number. The asymptotic lower bound of the Hamming weights of DBCs produced by any algorithm for n is linear $\frac{\log n}{8.25}$. This result changed the traditional idea that the asymptotic lower bound of the Hamming weight of a DBC produced by any algorithm may be sub-linear $\frac{\log n}{\log\log n}$. The time complexity and the space complexity of our dynamic programming algorithm to produce an optimal DBC were both the state-of-the-art. The recoding procedure of our algorithm was more than 20 times faster than Capuñay and Thériault's algorithm and more than 6 times faster than Bernstein, Chuengsatiansup, and Lange's algorithm.

Let $S(i)$ denote the smallest positive integer whose Hamming weight of its canonic DBCs is i. Our dynamic programming algorithm allowed us to find $S(i)$ for $i \leq 12$ immediately where $S(1) = 1$, $S(2) = 5$, $S(3) = 29$, $S(4) = 173$, $S(5) = 2093$, $S(6) = 14515$, $S(7) = 87091$, $S(8) = 597197$, $S(9) = 3583181$, $S(10) = 34936013$, $S(11) = 263363789$, and $S(12) = 1580182733$. This numerical fact provides a good impression about the sparseness of DBCs.

The cost function in this study was associated with $P+Q$, $2P$, and $3P$ for scalar multiplications. A direct promotion of the cost function is defined by $P+Q$, $P-Q$, $2P$, $2P+Q$, $3P$, and $3P+Q$. As the cost function is defined more precisely, an optimal DBC will improve scalar multiplications more. The optimal DBC can be directly generalized to a DBC with a large coefficient set of integers. Algorithm 1 can be generated to calculate the number of triple-base chains, and Algorithms $2-4$ can be extended to produce optimal extended DBCs and optimal triple-base chains.

Acknowledgments

The authors would like to thank the anonymous reviewers for many helpful comments and thank Guangwu Xu, Kunpeng Wang, Song Tian and Bei Liang for their helpful suggestions, especially for Guangwu Xu's suggestions on the parts of "Abstract" and "Introduction". This work is supported by the National Natural Science Foundation of China (Grants 61872442, 61502487, and 61772515) and the National Cryptography Development Fund (No. MMJJ20180216). W. Yu is supported by China Scholarship Council (No. 201804910201).

References

1. Dimitrov V., Imbert L., Mishra P.K.: Efficient and secure elliptic curve point multiplication using double-base chains, *ASIACRYPT 2005*, LNCS 3788, Springer, pp. 59-78, 2005. 1, 3, 2.2, 1, 3, 4, 5, 6.4, 6, 7, 8
2. Ciet M., Joye M., Lauter K., Montgomery P.L.: Trading inversions for multiplications in elliptic curve cryptography, *Designs, codes and cryptography* 39(6), pp. 189-206, 2006. 1, 3, 4.2, 5, 7, 8
3. Doche C., Habsieger L.: A tree-based approach for computing double-base chains, *ACISP 2008*, LNCS 5107, Springer, pp. 433-446, 2008. 1, 2, 3, 4.2, 4.2, 5, 7, 8
4. Imbert L., Philippe F.: strictly chained $(p,q)-$ary partitions, *Contibutions to Discrete Matheimatics 2010*, pp.119-136, 2010. 1, 3, 3.1, 3.1
5. Lou T., Sun X., Tartary C.: Bounds and trade-offs for double-base number systems, *Information Processing Letters*, vol. 111, no. 10, pp. 488-493, 2011. 1, 4.2, 4.2
6. Zhao C.A., Zhang F.G., Huang J.W.: Efficient Tate pairing computation using double-base chains, Sci. China Ser. F 51 , no. 8, pp. 1096-1105, 2008. 1, 3, 5.3
7. Capuñay A., Thériault N.: Computing optimal 2-3 chains for pairings. *LATINCRYPT 2015*, Springer, volume 9230, pp. 225-244, 2015. 1, 3, 4, 2.2, 3.1, 3.1, 4.1, 5, 6.3, 4, 5
8. Doche C., Kohel D., Sica F.: Double base number system for multi-scalar multiplications, *EUROCRYPT 2009*, LNCS 5479, Springer, pp. 502-519, 2009. 1
9. Adikari J., Dimitrov V.S., Imbert L.: Hybrid binary ternary number system for elliptic curve cryptosystems, *IEEE Trans. Computers*, vol. 60, pp. 254-265, Feb. 2011. 1
10. Doche C., Sutantyo D.: New and improved methods to analyze and compute double-scalar multiplications, *IEEE Trans. Computers*, 63(1), pp. 230-242, 2014. 1
11. Avanzi R.M., Dimitrov V.S., Doche C., Sica F.: Extending scalar multiplication using double bases. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 130-144. Springer, Heidelberg, 2006. 1
12. Mishra P.K., Dimitrov V.S.: Efficient quintuple formulas for elliptic curves and efficient scalar multiplication using multibase number representation, *ISC 2007*, Springer-Verlag, volume 4779, pp. 390-406, 2007. 1
13. Longa P., Gebotys C.: Fast multibase methods and other several optimizations for elliptic curve scalar multiplication, *PKC 2009: Proceedings of Public Key Cryptography*, LNCS 5443, Springer, pp. 443-462, 2009. 1, 5, 7, 8
14. Yu W., Wang K., Li B., Tian S.: Triple-base number system for scalar multiplications, *AFRICACRYPT 2013*, LNCS 7918, Springer, pp. 433-451, 2013. 1
15. Dimitrov V.S., Imbert L., Mishra P.K.: The double-base number system and its application to elliptic curve cryptography, *Math. Comp.* 77, no. 262, pp. 1075-1104, 2008. 1, 2, 3
16. Doche C.: On the enumeration of double-base chains with applications to elliptic curve cryptography, *ASIACRYPT 2014*, LNCS 8873, Springer, pp. 297-316, 2014. 1, 1, 3, 2.2, 3, 4.1, 4.1, 5, 6.3, 4

17. Bernstein D.J., Chuengsatiansup C., Lange T.: Double-base scalar multiplication revisited. http://eprint.iacr.org/2017/037 1, 3, 4, 2.2, 4.1, 5, 6, 3, 6.3, 4, 5
18. Cohen H., Miyaji A., Ono T.: Efficient elliptic curve exponentiation using mixed coordinates, *ASIACRYPT 1998*, LNCS 1514, Springer, pp.51-65, 1998. 2.1
19. Chevallier-Mames B., Ciet M., Joye M.: Low-cost solutions for preventing simple side-channel analysis: side-channel atomicity, *IEEE Trans. Computers*, vol. 53, no. 6, pp. 760-768, June 2004. 2.1
20. Longa P., Miri A.: Fast and flexible elliptic curve point arithmetic over prime fields, *IEEE Trans. Computers*, VOL. 57, NO. 3, pp. 289-302, March 2008. 2.1
21. Bernstein D.J., Lange T.: Explicit-formulas database, http://www.hyperelliptic.org/EFD/ 2.1
22. Renes J., Costello C., Batina L.: Complete addition formulas for prime order elliptic curves. *Advances in Cryptology - EUROCRYPT 2016*, Springer, pp. 403-428. 2016. 2.1
23. Meloni N., Hasan M.: Elliptic curve scalar multiplication combining Yao's algorithm and double Bases. *CHES 2009*, LNCS 5747, Springer, pp. 304-316, 2009. 2.2
24. Meloni N., Hasan M.: Efficient double bases for scalar multiplication. *IEEE Trans. Computers*, 64(8), pp. 2204-2212, 2015. 2.2
25. Disanto F., Imbert L., Philippe F.: On the maximal weight of (p,q)-ary chain partitions with bounded parts. https://www.emis.de/journals/INTEGERS/vol14.html 4
26. Yu W., Musa S., and Li B.: Double-base chains for scalar multiplications on elliptic curves. http://eprint.iacr.org/2020/144 3.1
27. Scott M.: MIRACL-multiprecision integer and rational arithmetic cryptographic library, C/C++ Library, ftp://ftp.computing.dcu.ie/pub/crypto/miracl.zip 3.2
28. Chalermsook P., Imai H., Suppakitpaisarn V.: Two lower bounds for shortest double-base number system. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol.E98-A, No.6, pp.1310-1312, 2015 4
29. Dimitrov V.S., Howe E.W.: Lower bounds on the lengths of double-base representations, *Proceedings of the American mathematical society*, vol. 139, Number 10, October, pp.3423-3430, 2011. 4.2
30. Kolmogorov A.N.: On tables of random numbers. *Theoretical Computer Science* 207, pp. 387-395, 1998. 4.2
31. Doche C., Imbert L.: Extended double-base number system with applications to elliptic curve cryptography. *INDOCRYPT* 2006, LNCS 4329, Springer, pp. 335-348, 2006. 4.2
32. Erdös P., Loxton J.H.: Some problems in partitio numerorum. *J. Austral. Math. Soc. Ser.* A 27(3), pp. 319-331, 1979. 5
33. Cormen T.H., Leiserson C.E., Rivest R.L., Stein C.: Introduction to algorithms, third edition. The MIT Press, Cambridge, Massachusetts London, England, 2009. 5, 5.1
34. Cook S.A.: On the minimum computation time of functions, Ph.D. thesis, Depatment of Mathematics, Harvard University, 1966. URL: https://cr.yp.to/bib/1966/cook.html. 6