Algorithmen & mathematische Modellierung

Forschungsschwerpunkt

Algorithmen und mathematische Modellierung

TU Graz
Graz University of Technology

Complements and Signed Digit Representations: Analysis of a Multi-Exponentiation-Algorithm of Wu, Lou, Lai and Chang

Clemens Heuberger and Helmut Prodinger

Project Area(s):
Analysis of Digital Expansions with Applications in Cryptography

Institut für Optimierung und Diskrete Mathematik (Math B)

Report 2008-8, April 2008

COMPLEMENTS AND SIGNED DIGIT REPRESENTATIONS: ANALYSIS OF A MULTI-EXPONENTIATION-ALGORITHM OF WU, LOU, LAI AND CHANG

CLEMENS HEUBERGER AND HELMUT PRODINGER

Abstract. Wu, Lou, Lai and Chang proposed a multi-exponentiation algorithm using binary complements and the non-adjacent form. The purpose of this paper is to show that neither the analysis of the algorithm given by its original proposers nor that by other authors are correct. In fact it turns out that the complement operation does not have significant influence on the performance of the algorithm and can therefore be omitted.

1. Introduction

An efficient way to compute a power a^n is to use the binary expansion $\sum_j d_j 2^j$ of n and compute a^n by a square and multiply algorithm [10],

$$a^{\sum_{j=0}^{\ell-1} d_j 2^j} = (((a^{d_{\ell}-1})^2 \cdot a^{d_{\ell-2}})^2 \cdots a^{d_1})^2 a^{d_0},$$

where the number of squarings needed is $\ell - 1$, whereas the number of multiplications by a^{d_j} equals to the number of nonzero d_j minus 1 (under the assumption that $d_{\ell-1} = 1$), because multiplications with a^0 can be omitted. Among the various possible optimisations is the use of signed digit representations, i.e., allowing digits -1 also, which results in multiplications by a^{-1}. This is of particular interest if a^{-1} is known or can be computed easily, e.g., in the point group of an elliptic curve.

Some cryptosystems also need multi-exponentiation $\prod_{j=1}^{D} a_j^{n_j}$ (usually for $D \in \{2, 3\}$). A trivial approach would be to compute $a_j^{n_j}$ separately for $j \in \{1, \ldots, D\}$ and multiply the results, however, Straus[1] [15] demonstrated that an interleaved approach leads to better results. For simplicity of exposition, we restrict ourselves to $D = 2$ at this point, although

2000 *Mathematics Subject Classification.* 11A63; 68W40 94A60.

Key words and phrases. Signed digit representation; Multi-exponentiation; Complement; Non-Adjacent-Form; Canonical signed digit representation.

This paper was written while C. Heuberger was a visitor at the Center of Experimental Mathematics at the University of Stellenbosch. He thanks the center for its hospitality. He is also supported by the Austrian Science Foundation FWF, project S9606, that is part of the Austrian National Research Network "Analytic Combinatorics and Probabilistic Number Theory."

H. Prodinger is supported by the NRF grant 2053748 of the South African National Research Foundation and by the Center of Experimental Mathematics of the University of Stellenbosch.

[1]This approach is frequently called Shamir's trick, we refer to [2] for a discussion of this attribution. Similar suggestions have been made in [11] and [4].

the method works for arbitrary D. For computing $a^m b^n$, we take binary expansions $m = \sum_{j=0}^{\ell-1} c_j 2^j$ and $n = \sum_{j=0}^{\ell-1} d_j 2^j$ and compute

$$a^{\sum_{j=0}^{\ell-1} c_j 2^j} b^{\sum_{j=0}^{\ell-1} d_j 2^j} = (((a^{c_{\ell-1}} b^{d_{\ell-1}})^2 a^{c_{\ell-2}} b^{d_{\ell-2}})^2 \dots a^{c_1} b^{d_1})^2 a^{c_0} b^{d_0}.$$

If $a^c b^d$ are precomputed for all admissible pairs of digits (c, d), then the number of squarings equals $\ell - 1$ and the number of multiplications by $a^{c_j} b^{d_j}$ equals the joint Hamming weight, i.e., the number of pairs $(c_j, d_j) \neq (0, 0)$, minus one (under the assumption that $(c_{\ell-1}, d_{\ell-1}) \neq (0, 0)$) plus the time needed for the precomputation, which is clearly constant and does not depend on the length of the expansion. In dimension $D = 2$, pairs of integers can be identified with complex numbers as proposed in [11], but this is merely an other way to formulate the procedure. Allowing negative digits again, redundancy can be used to decrease the joint Hamming weight.

As in the case of dimension 1, there is a syntactic condition which yields expansions of minimal joint Hamming weight, cf. [14], [5], [12]. In dimension $D = 2$, it is shown that these optimal expansions have expected Hamming weight $(1/2)\ell + O(1)$, so that the total expected number of multiplications equals[2] $(3/2)\ell + O(1)$, cf. also [1] and [6]. We refer to [7] for a more detailed introduction with more references.

The authors of [17] present an alternative approach in dimension D involving the complement of the binary expansion and claim that the expected number of multiplications of their algorithm equals $1.304\ell + O(1)$. This was followed by [16] whose authors claim to correct the result [17] and that the same algorithm needs $1.471\ell + O(1)$ multiplications on average. The purpose of this note is to show that both results are incorrect. We explain why the result cannot be better than the above optimal joint expansions (Theorem 1), show that the algorithm essentially corresponds to taking the NAF for both arguments (Theorem 2) and give the correct expected number $(14/9)\ell + O(1) = 1.555\dots\ell + O(1)$ of multiplications (Theorem 3).

In Section 2, we collect notations and well-known results on digit expansions. Section 3 presents the algorithm proposed by [17], which is analysed in Section 4. Finally, in Section 5, we discuss where the errors in the probabilistic arguments of [17] and [16] lie.

2. Digit Expansions

2.1. **Digit Expansions of Integers.** A signed digit expansion of an integer n is a word $d_{\ell-1} \dots d_0$ over the alphabet $\{-1, 0, 1\}$ such that $n = \mathsf{value}(d_{\ell-1} \dots d_0) = \sum_{j=0}^{\ell-1} d_j 2^j$. The *(Hamming) weight* $\mathsf{weight}(d_{\ell-1} \dots d_0)$ of $d_{\ell-1} \dots d_0$ is the number of non-zero digits d_j.

When all digits are in $\{0, 1\}$, we speak of the *standard binary expansion* of n (which must then be non-negative). The standard binary expansion of n is denoted by $\mathsf{Binary}(n)$.

While every integer n admits infinitely many signed digit expansions, one special expansion has attracted particular attention.

Definition 2.1. A signed digit expansion $d_{\ell-1} \dots d_0$ is called a *Non-Adjacent-Form (NAF)*, if $d_j d_{j+1} = 0$ for all j, i.e., there are no adjacent non-zero digits.

[2]The authors of [17] and [16] erroneously write 1.503ℓ without further comment.

Reitwiesner [13] showed that every integer n admits a unique NAF, denoted by $\mathsf{NAF}(n)$, and that $\mathsf{NAF}(n)$ minimises the Hamming weight over all signed digit expansions of n. The NAF is known under various names, e.g., the *canonical signed digit expansion.*

The ones' complement of a standard binary expansion $d_{\ell-1}\dots d_0$ is $\widehat{d_{\ell-1}}\dots\widehat{d_0}$, where

$$\widehat{d} = 1 - d = \begin{cases} 0 & d = 1, \\ 1 & d = 0. \end{cases}$$

It is immediate from the definition that

$$\mathsf{value}(d_{\ell-1}\dots d_0) = 2^\ell - \mathsf{value}(\widehat{d_{\ell-1}}\dots\widehat{d_0}) - 1.$$

This can be seen as another signed digit expansion,

$$\mathsf{value}(d_{\ell-1}\dots d_0) = \mathsf{value}(1(-\widehat{d_{\ell-1}})\dots(-\widehat{d_1})(-\widehat{d_0}-1)),$$

with the exception that the least significant digit is now in $\{-1,-2\}$.

2.2. **Digit Expansion of Vectors.** A signed digit joint expansion of an integer vector $\binom{m}{n}$ is a word $\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}$ over the alphabet $\{-1,0,1\}^2$ such that

$$\binom{m}{n} = \mathsf{value}(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}) = \sum_{j=0}^{\ell-1} \mathbf{d}_j 2^j.$$

The *joint (Hamming) weight* of $\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}$ is the number of j with $\mathbf{d}^{(j)} \neq \mathbf{0} := \binom{0}{0}$. The components of $\mathbf{d}^{(j)}$ are written as $\binom{d_1^{(j)}}{d_2^{(j)}}$.

So, the digits are now column vectors. One simple way to obtain such a joint expansion is to independently choose two signed-binary expansions $d_1^{(\ell-1)}\dots d_1^{(0)}$ and $d_2^{(\ell-1)}\dots d_2^{(0)}$ of m and n, respectively, and to write them on top of each other. In order to achieve small joint weight, one might take the NAFs of m and n and write them on top of each other; the expected joint weight is $(5/9)\ell + O(1)$, cf. [6].[3]

Solinas [14] discussed the "Joint Sparse Form", a joint expansion which minimises the joint weight over all joint expansions of the same pair of integers. Here, we use a simplified version introduced in [5].

Definition 2.2. A joint expansion $\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}$ is called a *Simple Joint Sparse Form (SJSF)*, if the following two conditions hold for all $j \geq 0$:

(1) If $|d_1^{(j)}| \neq |d_2^{(j)}|$, then $|d_1^{(j+1)}| = |d_2^{(j+1)}|$,

(2) If $|d_1^{(j)}| = |d_2^{(j)}| = 1$, then $\mathbf{d}^{(j+1)} = \mathbf{0}$.

[3]The heuristic argument to see this would be that the expected number of zeros in a NAF is $(2/3)\ell+O(1)$, thus the expected number of digits vectors $\mathbf{0}$ when writing two NAFs on top of each other should be $(4/9)\ell + O(1)$, which leaves $(5/9)\ell + O(1)$ for the Hamming weight. As we shall see when discussing the errors in [17] and [16], such arguments do not take possible dependence of the digits into account and might lead to errors. Therefore, we refer to the precise analysis in [6].

In [5], we proved that every pair of integers $\binom{m}{n}$ admits exactly one SJSF, denoted by $\mathsf{SJSF}\binom{m}{n}$, and that $\mathsf{SJSF}\binom{m}{n}$ minimises the joint weight over all joint expansions of $\binom{m}{n}$ with digits in $\{-1,0,1\}$. In [6], it was shown that the expected joint weight of a SJSF of length ℓ is $(1/2)\ell + O(1)$.

3. Multi-Exponentiation Algorithm

We now present the algorithm of [17] in Algorithm 1. We assume that $a_1^{d_1}a_2^{d_2}$ have been precomputed for $(d_1,d_2) \in \{-1,0,1\}^2$ and are used in Lines 13 and 15. It might happen that $d_k^{(0)} = -2$, in that case, two multiplications are needed in Line 15. Note that the length of the NAF may exceed the length of the standard binary expansion by at most 1.

Algorithm 1 Wu, Lou, Lai and Chang's [17] Algorithm for Multi-Exponentiation

Input: a_1, $a_2 \in G$ (some Abelian group), n_1, $n_2 \in \mathbb{N}$
Output: $b = a_1^{n_1}a_2^{n_2}$

1: $\ell = \lfloor \log_2(\max(n_1,n_2)) \rfloor + 1$
2: **for** $k = 1,2$ **do**
3: $\quad b_k^{(\ell-1)} \dots b_k^{(0)} \leftarrow \mathsf{Binary}(n_k)$
4: $\quad$ **if** $\mathsf{weight}(b_k^{(\ell-1)} \dots b_k^{(0)}) > \ell/2$ **then**
5: $\quad\quad (d_k^{(\ell)} \dots d_k^{(0)}) \leftarrow \mathsf{NAF}(\mathsf{value}((b_k^{(\ell-1)} - 1) \dots (b_k^{(0)} - 1)))$
6: $\quad\quad d_k^{(\ell)} \leftarrow d_k^{(\ell)} + 1$
7: $\quad\quad d_k^{(0)} \leftarrow d_k^{(0)} - 1$
8: $\quad$ **else**
9: $\quad\quad (d_k^{(\ell)} \dots d_k^{(0)}) \leftarrow \mathsf{NAF}(n_k)$
10: $\quad$ **end if**
11: $\quad$ {We have $n_k = \mathsf{value}(d_k^{(\ell)} \dots d_k^{(0)})$}
12: **end for**
13: $b \leftarrow a_1^{d_1^{(\ell)}} a_2^{d_2^{(\ell)}}$
14: **for** $j = \ell - 1$ **downto** 0 **do**
15: $\quad b \leftarrow b^2 a_1^{d_1^{(j)}} a_2^{d_2^{(j)}}$
16: $\quad$ {We have $b = a_1^{\sum_{k=j}^{\ell} d_1^{(k)} 2^{k-j}} a_2^{\sum_{k=j}^{\ell} d_2^{(k)} 2^{k-j}}$}
17: **end for**

The idea of the algorithm is the following: If the weight of the binary expansion of the exponent n_j is large ($> \ell/2$), then n_j is represented by its complement and the NAF of the complement is used. The heuristic is that reducing the weight of the expansion before converting it to its NAF should result in a lower weight of the NAF.

Note that after execution of Line 12 of Algorithm 1, we have a joint expansion $\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)}$ of $\mathbf{n} = \binom{n_1}{n_2}$ where $\mathbf{d}^{(j)} \in \{-1,0,1\}^2$ for $j > 0$ and $\mathbf{d}^{(0)} \in \{-2,-1,0,1\}^2$. The number of

group multiplications is ℓ (for the squarings) plus

$$\mathsf{weight}_1(\mathbf{d}^{(\ell)}\dots\mathbf{d}^{(0)}) := \sum_{j=0}^{\ell} \max\{|d_k^{(j)}| : k \in \{1,2\}\}$$

minus 1 (no multiplication is required for the most significant digit). Note that for expansions with digits $\{-1,0,1\}$, the notions of weight_1 and weight agree.

4. Analysis of the Algorithm

We will now extend the optimality proof for the SJSF from [5] to the case of digits from $\{-2,-1,0,1,2\}$. It turns out that we can allow arbitrary digits of absolute value at most 2 without changing the result. In fact, we do not even need any particular properties of the SJSF.

Theorem 1. *Let* $\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}$ *be a word over the alphabet* $\{-2,-1,0,1,2\}^2$ *and* $\mathbf{n} = \mathsf{value}(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)})$. *Then we have*

$$\mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}) \geq \mathsf{weight}_1(\mathsf{SJSF}(\mathbf{n})) = \mathsf{weight}(\mathsf{SJSF}(\mathbf{n})),$$

i.e., $\mathsf{SJSF}(\mathbf{n})$ *minimises* weight_1 *over all expansions of* $\mathbf{n}$ *with digits in* $\{-2,-1,0,1,2\}$.

Before proving the theorem, we note that this already shows that the analysis in [17] and [16] cannot be correct:

Corollary 4.1. The expected number of group multiplications needed by Algorithm 1 is at least $(3/2)\ell + O(1)$.

Proof of Corollary 4.1. For every $\mathbf{n}$, the number of multiplications used by Algorithm 1 is not less then the number of multiplications needed when using the SJSF, which is known to be $(3/2)\ell + O(1)$ from [6]. □

The essential step in the proof of Theorem 1 is the following lemma.

Lemma 4.2. *Let* $\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}$ *be a word over the alphabet* $\{-2,-1,0,1,2\}^2$ *where* $k > 0$ *digit vectors contain a digit of absolute value* 2. *Then there is a word* $\mathbf{c}^{(\ell'-1)}\dots\mathbf{c}^{(0)}$ *over the alphabet* $\{-2,-1,0,1,2\}^2$ *with less than* k *digit vectors containing a digit of absolute value* 2, $\mathsf{value}(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}) = \mathsf{value}(\mathbf{c}^{(\ell'-1)}\dots\mathbf{c}^{(0)})$ *and* $\mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}) \geq \mathsf{weight}_1(\mathbf{c}^{(\ell'-1)}\dots\mathbf{c}^{(0)})$.

Proof. We prove the lemma by induction on $\mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)})$. Choose j maximal such that $\mathbf{d}^{(j)}$ contains a digit of absolute value 2. We write $\mathbf{d}^{(j)} = 2\mathbf{q} + \mathbf{r}$ with $\mathbf{q} \in \{-1,0,1\}^2$ and $\mathbf{r} \in \{0,1\}^2$. We have $\mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(j+2)}(\mathbf{d}^{(j+1)} + \mathbf{q})) \leq \mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}) - 1$ and $\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(j+2)}(\mathbf{d}^{(j+1)} + \mathbf{q})$ is an expansion with digits from $\{-2,-1,0,1,2\}$, where digits of absolute value 2 can only occur in $(\mathbf{d}^{(j+1)} + \mathbf{q})$. Thus, by induction hypothesis, there is an expansion $\mathbf{c}^{(\ell'-1)}\dots\mathbf{c}^{(j+1)}$ with digits from $\{-1,0,1\}$, $\mathsf{value}(\mathbf{c}^{(\ell'-1)}\dots\mathbf{c}^{(j+1)}) = \mathsf{value}(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(j+2)}(\mathbf{d}^{(j+1)} + \mathbf{q}))$ and

$$\mathsf{weight}_1(\mathbf{c}^{(\ell'-1)}\dots\mathbf{c}^{(j+1)}) \leq \mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(j+2)}(\mathbf{d}^{(j+1)} + \mathbf{q})).$$

Setting $\mathbf{c}^{(j)}\mathbf{c}^{(j-1)}\dots\mathbf{c}^{(0)} = \mathbf{r}\mathbf{d}^{(j-1)}\dots\mathbf{d}^{(0)}$, we see that

$$\begin{aligned}\mathsf{weight}_1(\mathbf{c}^{(\ell'-1)}\dots\mathbf{c}^{(0)}) &\le \mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(j+2)}(\mathbf{d}^{(j+1)}+\mathbf{q})) + \mathsf{weight}_1(\mathbf{r}\mathbf{d}^{(j-1)}\dots\mathbf{d}^{(0)})\\ &\le \mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(j+2)}\mathbf{d}^{(j+1)}) + 1 + 1 + \mathsf{weight}_1(\mathbf{d}^{(j-1)}\dots\mathbf{d}^{(0)})\\ &= \mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)})\end{aligned}$$

and that $\mathbf{c}^{(\ell'-1)}\dots\mathbf{c}^{(0)}$ satisfies the requirements of the lemma. □

We are now able to prove Theorem 1.

Proof of Theorem 1. Repeated application of Lemma 4.2 shows that there is an expansion $\mathbf{c}^{(\ell'-1)}\dots\mathbf{c}^{(0)}$ with digits from $\{-1,0,1\}$ with $\mathsf{value}(\mathbf{c}^{(\ell-1)}\dots\mathbf{c}^{(0)}) = \mathbf{n}$ and

$$\mathsf{weight}_1(\mathbf{d}^{(\ell-1)}\dots\mathbf{d}^{(0)}) \ge \mathsf{weight}_1(\mathbf{c}^{(\ell-1)}\dots\mathbf{c}^{(0)}).$$

Taking into account that $\mathsf{weight}_1(\mathbf{c}^{(\ell-1)}\dots\mathbf{c}^{(0)}) = \mathsf{weight}(\mathbf{c}^{(\ell-1)}\dots\mathbf{c}^{(0)})$ and the optimality of the SJSF [5, Theorem 2] completes the proof of the theorem. □

The next question is how Algorithm 1 compares with the simple strategy of directly using $\binom{\mathsf{NAF}(n_1)}{\mathsf{NAF}(n_2)}$.

Theorem 2. *Let $b_{\ell-1}\dots b_0$ be a standard binary expansion and $\widehat{b_{\ell-1}}\dots\widehat{b_0}$ its complement. Then*

$$|\mathsf{weight}(\mathsf{NAF}(\mathsf{value}(b_{\ell-1}\dots b_0))) - \mathsf{weight}(\mathsf{NAF}(\mathsf{value}(\widehat{b_{\ell-1}}\dots\widehat{b_0})))| \le 2.$$

This means that the strategy of taking the NAF of the complement of a number at best induces a saving of 2 in the weight, which is subsequently lost when adding the two corrective terms. In other words, this strategy never yields a lower weight than a direct use of the NAF.

Proof of Theorem 2. The NAF can be computed from the standard binary expansion of a positive integer n by a transducer automaton from right to left (cf. [8, Figure 2]), reproduced here as Figure 1. For typographical reasons, negative digits $-d$ are written as $\bar{d}$.

In order to compare the NAFs of n and its complement, we compute these NAFs simultaneously by one transducer. This transducer is shown in Figure 2. Here, $\perp$ denotes the end of the input and ε denotes the empty word.

The transducer reads the standard binary expansion of n from right to left and writes vectors of digits containing the NAF of n and its complement. The labels of the states correspond to carries, the "binary point" indicates the look-ahead, i.e., the number of digits read minus the number of digits written. The transducer can be decomposed in four strongly connected components: $\mathcal{C}_1 = \{\binom{0}{0}\}$ (the initial state only), $\mathcal{C}_2 = \{\cdot\binom{0}{1}, \cdot\binom{1}{0}\}$, $\mathcal{C}_3 = \{\cdot\binom{0}{2}, \cdot\binom{1}{1}, \cdot\binom{2}{0}\}$ and $\mathcal{C}_4 = \{\text{terminal state}\}$.

When leaving $\mathcal{C}_1$, the weights of the two output rows are trivially equal. After leaving $\mathcal{C}_2$, the difference of the weights is at most 1, as there is only one cycle in $\mathcal{C}_2$ and its weights are balanced. Within $\mathcal{C}_3$, the weight of the output in both rows is always the same. When

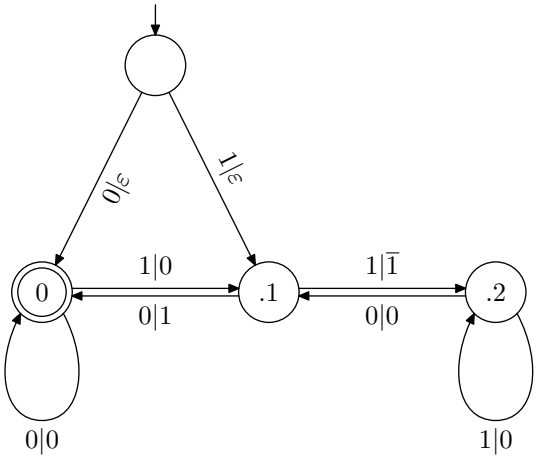

FIGURE 1. Transducer for converting the standard binary expansion to the NAF.

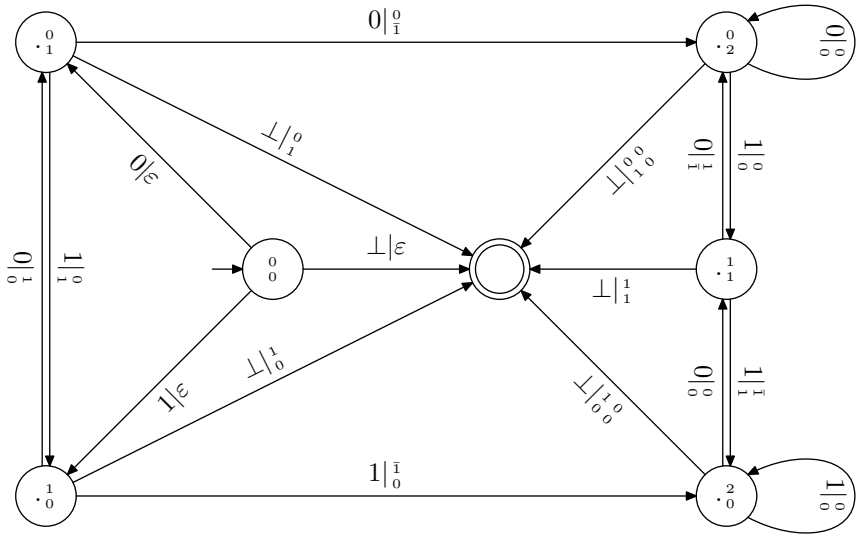

FIGURE 2. Transducer converting a standard binary expansion $d_{\ell-1}\dots d_0$ into $\begin{pmatrix}\mathsf{NAF}(\mathsf{value}(d_{\ell-1}\dots d_0))\\ \mathsf{NAF}(\mathsf{value}(\widehat{d_{\ell-1}}\dots\widehat{d_0}))\end{pmatrix}$, i.e., the NAF and the NAF of the complement.

leaving $\mathcal{C}_3$, another difference of at most 1 might occur. Summing up, the weight difference is at most 2. □

We can now quantify the performance of Algorithm 1:

Theorem 3. *For a random pair (n_1, n_2) with $0 \le n_1, n_2 < 2^\ell$ (all of these pairs are considered to be equally likely), the expected number of multiplications when executing Algorithm 1 is*

$$\frac{14}{9}\ell + O(1) = 1.555\dots\ell + O(1).$$

Proof. Before considering pairs, it is essential to understand the effect of Algorithm 1 on a single integer, which is encoded by the transducer automaton in Figure 2. We number the states of the transducer as follows:

number	1	2	3	4	5	6
state	$\begin{smallmatrix}0\\0\end{smallmatrix}$	$\begin{smallmatrix}0\\\cdot 1\end{smallmatrix}$	$\begin{smallmatrix}1\\\cdot 0\end{smallmatrix}$	$\begin{smallmatrix}0\\\cdot 2\end{smallmatrix}$	$\begin{smallmatrix}2\\\cdot 0\end{smallmatrix}$	$\begin{smallmatrix}1\\\cdot 1\end{smallmatrix}$

The transition probability matrix of the transducer is

$$P = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix},$$

i.e., the entry in row i, column j, is the probability of a transition from state i to state j. These are 0 or $1/2$ depending on whether there is a transition from i to j at all; the digits of the standard binary expansion are independently uniformly distributed.

The probability of reaching state j after reading k digits is the jth component of

$$(1,0,0,0,0,0)P^k = \left(0, 2^{-k}, 2^{-k}, \frac{1}{3} + O(2^{-k}), \frac{1}{3} + O(2^{-k}), \frac{1}{3} + O(2^{-k})\right).$$

When leaving States 4 or 5, the transducer writes a digit 0, when leaving State 6, a non-zero digit is written. The situation in States 2 and 3 is more complicated as it depends on the weight of the standard binary expansion, however, since we are in these states with probability 2^{-k}, we do not have to deal with this problem. Summing up, the probability that the transducer writes a digit 0 as the $(k-1)$st output digit is $p_{k-2} := 2/3 + O(2^{-k})$.

Let now $\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)}$ be the joint expansion of (n_1, n_2) produced by Algorithm 1, where n_1 and n_2 are independent and uniformly distributed random variables on $\{0, \dots, 2^\ell - 1\}$. Denote by $\mathsf{zeros}(\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)})$ the number of digit vectors $\mathbf{0}$ in $\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)}$, which implies that $\mathsf{zeros}(\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)}) = \ell + 1 - \mathsf{weight}(\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)})$. Then the expectation of $\mathsf{zeros}(\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)})$ can be computed as

$$\begin{aligned} \mathbb{E}\left(\mathsf{zeros}(\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)})\right) &= \sum_{k=0}^{\ell} \mathbb{P}\left(\mathbf{d}^{(k)} = \mathbf{0}\right) = \sum_{k=0}^{\ell} \mathbb{P}\left(d_1^{(k)} = 0\right) \mathbb{P}\left(d_2^{(k)} = 0\right) \\ &= \sum_{k=0}^{\ell} p_k^2 = \sum_{k=0}^{\ell} \left(\frac{4}{9} + O(2^{-k})\right) = \frac{4}{9}\ell + O(1), \end{aligned}$$

where we used the fact that the random variables $d_1^{(k)}$ and $d_2^{(k)}$ are independent (which is a consequence of the fact that n_1 and n_2 have been assumed to be independent). From this, we see that

$$\mathbb{E}\left(\mathsf{weight}(\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)})\right) = \ell + 1 - \mathbb{E}\left(\mathsf{zeros}(\mathbf{d}^{(\ell)} \dots \mathbf{d}^{(0)})\right) = \frac{5}{9}\ell + O(1),$$

and the results follows by adding the unavoidable ℓ squarings. □

Remark 4.3. This proof can easily be generalised to higher dimensions. In dimension D, we obtain

$$\mathbb{E}\left(\mathsf{weight}(\mathbf{d}^{(\ell)}\dots\mathbf{d}^{(0)})\right) = \left(1-\left(\frac{2}{3}\right)^{D}\right)\ell + O(1).$$

On the other hand, the approach in [6] can be generalised to explicitly give the constants now hidden in the error term at the cost of more complicated transducers and a delicate analysis of the influence of the weight of the standard binary expansion, cf. [9].

5. Errors in [17] and [16]

The purpose of this section is to point out where the errors in [17] and [16] occurred. The authors of [17] write on page 1072 in Section 4:

> "Besides, the average proportion of non-zeros in binary representation is $\frac{1}{2}$ and in canonical-signed-digit binary representation is $\frac{1}{3}$. So the average proportion of zeros in the proposed algorithm is $(1-\frac{1}{2}\times\frac{1}{3}) = \frac{5}{6}$."

This assertion is erroneous, because the multiplication of 1/2 and 1/3 cannot be justified in any way: The weight of a NAF is roughly 1/3 times the length of the expansion, not 1/3 times the weight of the standard binary expansion. Moreover, the weights of the NAF and the standard binary expansion are only asymptotically independent, cf. [9].

The authors of [16] write on page 1851:

> "Before the complement recoding, each bit of $E = (e_{k-1}\dots e_1e_0)_2$ assumes a value of 0 or 1 with equal probability, i.e. $P(e_i = 0) = P(e_i = 1) = 1/2$ for $0 \le i \le k-1$, and there is no dependency between any two bits. After the complement recoding, it should be a value of 0 or 1 with unequal probability, i.e. $P(e_i = 0) = 3/4$ and $P(e_i = 1) = 1/4$ for $0 \le i \le k-1$. Certainly, there is still no dependency between any two bits."

By construction, there *is* some dependence between the bits, as at most half of them can be equal to 1. Next, the claimed probabilities 3/4 and 1/4 are incorrect, this has also been discussed in detail by Yen, Lien and Moon [18] while correcting erroneous claims in [3].

6. Conclusions

We analysed the multi-exponentiation algorithm from [17]. It turns out that the performance estimates in [17] and [16] are based on incorrect probabilistic assumptions and are wrong. The method due to Solinas [14] or its equivalent formulations have better performance. Even worse, taking the complement does not have any positive effect, because the non-adjacent form essentially ignores the influence of the complement. Thus the proposed algorithm performs as if one would simply take the NAF for both arguments, which corresponds to the method proposed by [4] (without improvements) and is known to be not optimal.

References

[1] R. M. Avanzi, *The complexity of certain multi-exponentiation techniques in cryptography*, J. Cryptology **18** (2005), 357–373.

[2] D. Bernstein, *Pippenger's exponentiation algorithm*, Preprint. Available at `http://cr.yp.to/papers.html`, 2002.

[3] Chin-Chen Chang, Ying-Tse Kuo, and Chu-Hsing Lin, *Fast algorithms for common-multiplicand multiplication and exponentiation by performing complements*, 17th International Conference on Advanced Information Networking and Applications, 2003. AINA 2003, 2003, pp. 807–811.

[4] V. S. Dimitrov, G. A. Jullien, and W. C. Miller, *Complexity and fast algorithms for multiexponentiations*, IEEE Trans. Comput. **49** (2000), 141–147.

[5] P. J. Grabner, C. Heuberger, and H. Prodinger, *Distribution results for low-weight binary representations for pairs of integers*, Theoret. Comput. Sci. **319** (2004), 307–331.

[6] P. J. Grabner, C. Heuberger, H. Prodinger, and J. Thuswaldner, *Analysis of linear combination algorithms in cryptography*, ACM Trans. Algorithms **1** (2005), 123–142.

[7] C. Heuberger and J. Muir, *Minimal weight and colexicographically minimal integer representations*, J. Math. Cryptol. **1** (2007), 297–328.

[8] C. Heuberger and H. Prodinger, *Analysis of alternative digit sets for nonadjacent representations*, Monatsh. Math. **147** (2006), 219–248.

[9] ———, *The Hamming weight of the Non-Adjacent-Form under various input statistics*, Period. Math. Hungar. **55** (2007), 81–96.

[10] D. E. Knuth, *Seminumerical algorithms*, third ed., The Art of Computer Programming, vol. 2, Addison-Wesley, 1998.

[11] K. Z. Pekmestzi, *Complex number multipliers*, IEE Proceedings — Computers and Digital Techniques **136** (1989), 70–75.

[12] J. Proos, *Joint sparse forms and generating zero columns when combing*, Tech. Report CORR 2003-23, Centre for Applied Cryptographic Research, University of Waterloo, 2003, available at `http://www.cacr.math.uwaterloo.ca/techreports/2003/corr2003-23.ps`.

[13] G. W. Reitwiesner, *Binary arithmetic*, Advances in computers, vol. 1, Academic Press, New York, 1960, pp. 231–308.

[14] J. A. Solinas, *Low-weight binary representations for pairs of integers*, Tech. Report CORR 2001-41, Centre for Applied Cryptographic Research, University of Waterloo, 2001, available at `http://www.cacr.math.uwaterloo.ca/techreports/2001/corr2001-41.ps`.

[15] E. Straus, *Addition chains of vectors (Problem 5125)*, Amer. Math. Monthly **71** (1964), 806–808.

[16] Da-Zhi Sun, Jin-Peng Huai, Ji-Zhou Sun, and Jia-Wan Zhang, *Computational efficiency analysis of Wu et al.'s fast modular multi-exponentiation algorithm*, Appl. Math. Comput. **190** (2007), 1848–1854.

[17] Chia-Long Wu, Der-Chyuan Lou, Jui-Chang Lai, and Te-Jen Chang, *Fast modular multi-exponentiation using modified complex arithmetic*, Appl. Math. Comput. **186** (2007), 1065–1074.

[18] Sung-Ming Yen, Wei-Chih Lien, and SangJae Moon, *Inefficiency of common-multiplicand multiplication and exponentiation algorithms by performing binary complements*, Appl. Math. Comput. **189** (2007), 285–290.

Institut für Mathematik B, Technische Universität Graz, Austria
E-mail address: `clemens.heuberger@tugraz.at`

Department of Mathematics, University of Stellenbosch, South Africa
E-mail address: `hproding@sun.ac.za`