THE ALTERNATING GREEDY EXPANSION AND APPLICATIONS TO LEFT-TO-RIGHT ALGORITHMS IN CRYPTOGRAPHY

CLEMENS HEUBERGER‡, RAJENDRA KATTI, HELMUT PRODINGER*, AND XIAOYU RUAN

ABSTRACT. In [4], we introduced the alternating greedy expansion of integers, which turned out to be useful in several left-to-right algorihms in cryptography. In this paper, we collect known results about this alternating greedy expansion and complement it with other useful properties and algorithms. In the second part, we apply it to give an algorithm for computing a joint expansion of d integers of minimal joint Hamming weight from left to right, i.e., from the column with the most significant bits towards the column with the least significant bits. Furthermore, we can also compute an expansion equivalent to the so-called w-NAF from left to right using the alternating greedy expansion.

1. Introduction

In [4], we introduced the "alternating greedy expansion" of integers. We proved that there is exactly one such expansion for each integer, we described a greedy algorithm for computing it as well as a transducer automaton which transforms the (unsigned) binary expansion to this alternating greedy expansion from left to right.

In that paper, it turned out that this alternating greedy expansion is useful for finding a joint signed binary expansion of two integers of minimal joint Hamming weight (the number of nonzero columns) from left to right. Such minimal expansions can be used in elliptic curve cryptography to accelerate the computation. Since the digits are used from left to right, we are interested in generating them from left to right, too, instead of storing the whole expansion after using a right-to-left algorithm. The minimality proof relied on a counting argument using generating functions, thus it was not very intuitive and cannot be used for general dimension d. Although we never stated it explicitly, the left-to-right algorithm for computing minimal signed binary expansions of single integers as discussed in [5] (cf. also Joye and Yen [7]) can also be seen to be based on the alternating greedy expansion.

In the meantime, we described a similar left-to-right algorithm for pairs of integers in [8]. Furthermore, a left-to-right analogue to the w-NAF of integers has been proposed by Avanzi [1]. This could also be formulated in terms of the alternating greedy expansion.

Since the alternating greedy expansion turned out to be useful in these contexts, it seems to be worth a more detailed study.

‡ This author is supported by the grant S8307-MAT of the Austrian Science Fund.

* This author is supported by the grant NRF 2053748 of the South African National Research Foundation.

The aim of the present paper is to study this alternating greedy expansion and its applications in more detail: We first discuss the alternating greedy expansion in Section 2. In particular, we give algorithms for computing the alternating greedy expansion from left to right or from right to left or in parallel from the digits of the unsigned binary expansion. This also yields a more detailed proof of uniqueness. We conclude that section by some estimates for alternating greedy expansions.

Sections 3 to 5 are devoted to the generalization of the above mentioned left-to-right algorithm for minimal joint expansions to arbitrary dimensions. To this aim, we review the known results and right-to-left algorithms in Section 3 and study the effect of taking alternating greedy expansions as an input. This enables us to describe an optimal left-to-right algorithm using the right-to-left algorithm as a subroutine in Section 4. The basic principle is to apply the right-to-left algorithm on a block of leading digits. Since the alternating greedy expansion blocks carries in a special way, this procedure is shown to be optimal. In Section 5 we refine the algorithm of Section 4 by avoiding superfluous replacements and by formulating it as a single algorithm from left to right.

Finally, we also consider Avanzi's [1] problem in Section 6 and show that the alternating greedy expansion gives a natural explanation in this situation, too. More precisely, we show that by using the alternating greedy expansion, it is easy to find an expansion (from left to right) with digits in the set $\{-(2^w-1), -(2^w-3), \dots, -3, -1, 0, 1, 3, \dots, 2^w-3, 2^w-1\}$ of minimal Hamming weight. The corresponding right-to-left algorithm yields the so-called w-NAF, i.e., the unique expansion with the same digits where all non-zeros are separated by at least w zeros.

2. The Alternating Greedy Expansion

Throughout this paper, a *(signed binary) expansion of an integer* n is an $\boldsymbol{\varepsilon} = (\varepsilon_j)_{j\in\mathbb{N}_0} = (\dots, \varepsilon_2, \varepsilon_1, \varepsilon_0) \in \{-1, 0, 1\}^{\mathbb{N}_0}$ such that only a finite number of ε_j is nonzero and $n = \mathsf{value}(\boldsymbol{\varepsilon}) := \sum_{j\geq 0} \varepsilon_j 2^j$. We will identify finite and (left) infinite sequences in a natural way by padding with leading zeros where appropriate.

In [4] we introduced the notion of an *alternating greedy expansion* of integers: An expansion $\boldsymbol{\varepsilon}$ of an integer n is called a *(balanced) alternating greedy expansion of* n, if

$$\text{if } \varepsilon_j = \varepsilon_\ell \neq 0 \text{ for some } j < \ell, \text{ then there is a } k \text{ with } j < k < \ell \text{ such that } \varepsilon_j = -\varepsilon_k = \varepsilon_\ell, \tag{1}$$

$$\text{for } \underline{j} := \min\{j : \varepsilon_j \neq 0\} \text{ and } \overline{j} := \max\{j : \varepsilon_j \neq 0\}, \text{ we have } \operatorname{sign}(n) = \varepsilon_{\overline{j}} = -\varepsilon_{\underline{j}}. \tag{2}$$

The existence of the alternating greedy expansion of an integer has been proved in [4] by a transducer automaton and we stated there that it is unique without giving a detailed proof. We give a detailed proof now by exhibiting the (algorithmic) bijection between the unsigned binary expansion of a positive integer n, i.e., the unique expansion of n with digits $\{0, 1\}$, to the alternating greedy expansion of n.

We therefore discuss the following two algorithms: Algorithm 1 computes the alternating greedy expansion of a positive integer from its unsigned binary expansion while Algorithm 2 computes the unsigned binary expansion from the alternating greedy expansion. Both algorithms operate from left to right.

Algorithm 1 algBINtoAGE: Computing the Alternating Greedy Expansion from the Unsigned Binary Expansion from Left to Right

---

Input: ℓ-bit unsigned binary expansion $(\eta_{\ell-1}, \eta_{\ell-2}, \ldots, \eta_1, \eta_0)$ of a positive integer n.
Output: The alternating greedy expansion $(\varepsilon_\ell, \varepsilon_{\ell-1}, \ldots, \varepsilon_1, \varepsilon_0)$ of n.
  $\eta_\ell \leftarrow 0$
  $\eta_{-1} \leftarrow 0$
  for $\ell \geq j \geq 0$ **do**
    $\varepsilon_j \leftarrow \eta_{j-1} - \eta_j$
  end for

Algorithm 1 can be realized by the transducer in Figure 1 working from left to right, i.e., reading the most significant digits first. In our transducers, we will write $\bar{1}$ for the digit -1. The label of the states corresponds to the last input digit read. The same transducer (with other labels for the states) has been shown in Figure 2 of [4].

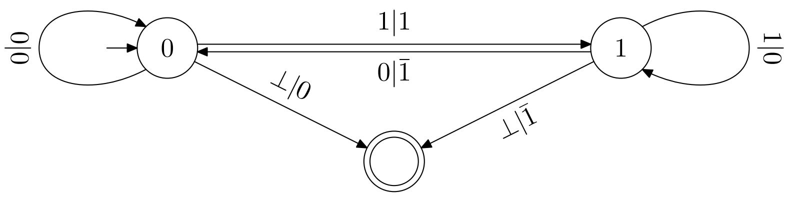

FIGURE 1. Transducer realizing algBINtoAGE from left to right. The symbol $\perp$ denotes the end of the sequence.

Algorithm 2 algAGEtoBIN: Computing the Unsigned Binary Expansion from the Alternating Greedy Expansion from Left to Right

---

Input: $(\ell+1)$-bit alternating greedy expansion $(\varepsilon_\ell, \varepsilon_{\ell-1}, \ldots, \varepsilon_1, \varepsilon_0)$ of a positive integer n
Output: The unsigned binary expansion $(\eta_{\ell-1}, \eta_{\ell-2}, \ldots, \eta_1, \eta_0)$ of n.
  $\eta_\ell \leftarrow 0$
  for $\ell - 1 \geq j \geq 0$ **do**
    $\eta_j \leftarrow \eta_{j+1} + \varepsilon_{j+1}$
  end for

Algorithm 2 can be realized by the transducer in Figure 2. The labels of the states correspond to the next output digit which has to be written (note that Algorithm 2 computes ε_j from the digits in position $(j+1)$).

We now reprove the existence and uniqueness of the alternating greedy expansion.

Theorem 1. *The alternating greedy expansion of a positive integer exists and is unique. The alternating greedy expansion of n can be computed by digitwise subtraction of the unsigned binary expansions of $2n$ and n.*

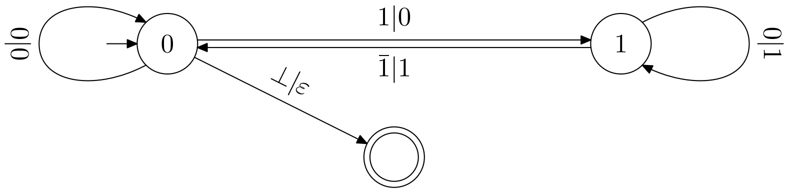

FIGURE 2. Transducer realizing algAGEtoBIN from left to right.

Proof. We note that Algorithm 1 corresponds to the transducer given in Figure 1 which implies that it is correct. This reproves the existence of the alternating greedy expansion.

On the other hand, Algorithm 2 is easily seen (by inspecting the transducer in Figure 2) to compute an unsigned binary expansion. We clearly see that the two algorithms are inverse to each other, which implies that they yield a bijection between the unsigned binary expansions and the alternating greedy expansions. This immediately implies uniqueness of the alternating greedy expansion and correctness of Algorithm 2.

Algorithm 1 shows that the alternating greedy expansion of n can be computed by digitwise subtraction of the unsigned binary expansions of $2n$ and n. □

We note that the representation of the alternating greedy expansion as digitwise difference of the binary expansions of $2n$ and n yields the explicit digit formula

$$\varepsilon_j = \eta_{j-1} - \eta_j = \left\lfloor \frac{n}{2^{j-1}} \right\rfloor - 3 \left\lfloor \frac{n}{2^j} \right\rfloor + 2 \left\lfloor \frac{n}{2^{j+1}} \right\rfloor .$$

where η_j and ε_j denote the digit in position j of the unsigned binary and the alternating greedy expansion of n, respectively. Several other digit expansions can be written as a digitwise linear combination of the unsigned binary expansion, cf. Prodinger [9]. Such representations enable one to perform a detailed analysis of the frequency of digits (or subblocks) as described in [2], for instance.

Obviously, the "for" loop in Algorithm 1 can be run for $j = \ell$ downto 0 as well as for $j = 0$ upto ℓ. Therefore, the alternating greedy expansion of a positive integer n can also be computed by a transducer from right to left, cf. Figure 3.

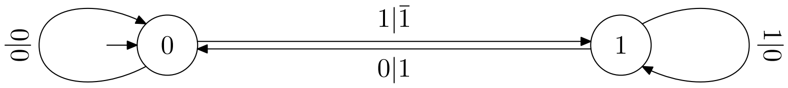

FIGURE 3. Transducer transforming the unsigned binary expansion to the alternating greedy expansion from right to left.

For later use, we collect bounds for the alternating greedy expansion as well as for the expansion satisfying (1), but violating (2). This latter expansion will also turn out to be useful, so we make the following definition: An expansion $\boldsymbol{\varepsilon}$ of n satisfying (1) and

$$\varepsilon_{\overline{j}} = \varepsilon_{\underline{j}} \text{ for } \underline{j} := \min\{j : \varepsilon_j \neq 0\} \text{ and } \overline{j} := \max\{j : \varepsilon_j \neq 0\}$$

will be called an *unbalanced alternating greedy expansion* of n.

Lemma 2. *Let $\boldsymbol{\varepsilon} \neq \mathbf{0}$ be an expansion of an integer n and $\ell = \max\{j : \varepsilon_j \neq 0\}$.*

(1) *We have* $\operatorname{sign}(n) = \operatorname{sign}(\varepsilon_\ell)$.
(2) *If $\boldsymbol{\varepsilon}$ is a balanced alternating greedy expansion, we have*
$$2^{\ell-1} \leq |n| < 2^{\ell}. \tag{3}$$
(3) *If $\boldsymbol{\varepsilon}$ is an unbalanced alternating greedy expansion, we have*
$$2^{\ell-1} < |n| \leq 2^{\ell}. \tag{4}$$

Proof. The first part follows from the fact that $\left|n - \varepsilon_\ell 2^\ell\right| < 2^\ell$. To prove the other two parts, we note that
$$|n| = \varepsilon_\ell n = 2^\ell - \left|\mathsf{value}(\varepsilon_{\ell-1}, \ldots, \varepsilon_0)\right|.$$
Since $(\varepsilon_{\ell-1}, \ldots, \varepsilon_0)$ is the alternating greedy expansion of the other type in each case (with most significant digit ε_k for some $k < \ell$), we get the estimates by induction. □

At this point we note that Lemma 2 also reproves the uniqueness of the alternating greedy expansions, since $\ell = \lfloor \log_2 |n| \rfloor + 1$ and $\varepsilon_\ell = \operatorname{sign}(n)$.

We also note that the digits of the unbalanced alternating greedy expansion can be calculated as digitwise subtraction and addition
$$2(n-1) - (n-1) + 1,$$
since $2(n-1) - (n-1)$ gives the balanced alternating greedy expansion of $(n-1)$ and the addition of $+1$ gives an expansion of n. Since the addition of $+1$ either cancels a -1 at the least significant position or replaces a least significant bit 0 by 1, the resulting expansion is indeed the unbalanced greedy expansion of n which has been proved to be unique.

3. The Simple Joint Sparse Form Revisited

For some $d \geq 1$, let $x^{(1)}, \ldots, x^{(d)}$ be integers. A *joint (signed binary) expansion of* $x^{(1)}, \ldots, x^{(d)}$ is an $\boldsymbol{\varepsilon} = (\varepsilon_j^{(k)})_{\substack{1 \leq k \leq d \\ j \in \mathbb{N}_0}} = (\boldsymbol{\varepsilon}_j)_{j \in \mathbb{N}_0} \in \{-1, 0, 1\}^{d \times \mathbb{N}_0}$ such that only a finite number of $\varepsilon_j^{(k)}$ is nonzero and such that $x^{(k)} = \sum_{j \geq 0} \varepsilon_j^{(k)} 2^j$ for $1 \leq k \leq d$. Its *joint Hamming weight* is the number of $j \geq 0$ such that $\boldsymbol{\varepsilon}_j \neq \mathbf{0}$. The $\boldsymbol{\varepsilon}_j$, $j \geq 0$, will be called the *columns* of the expansion.

We recall from Solinas [11] that a joint expansion $\boldsymbol{\varepsilon}$ of integers $x^{(1)}, \ldots, x^{(d)}$ can be used for computing the linear combination $x^{(1)} P_1 + \cdots + x^{(d)} P_d$ of d points $P_1, \ldots, P_d$ on an elliptic curve E, which is a frequent operation in elliptic curve cryptography. The idea is to define $Q_{\ell+1} := 0$ and $Q_j = 2Q_{j+1} + \sum_{i=1}^{d} \varepsilon_j^{(i)} P_i$. Then $x^{(1)} P_1 + \cdots + x^{(d)} P_d = Q_0$. We can precompute all sums $\sum_{i=1}^{d} s_i P_i$ for $(s_1, \ldots, s_d) \in \{-1, 0, 1\}^d$. Then the number of point additions equals the joint Hamming weight of $\boldsymbol{\varepsilon}$.

Solinas [11] proposed the so-called *joint sparse form* of two integers which minimizes the joint Hamming weight. In [3], we studied a simpler expansion and generalized the concept to higher dimensions. We proved that there is a unique joint expansion $\boldsymbol{\varepsilon}$, called the *simple joint sparse form*, of $x^{(1)}, \ldots, x^{(d)}$ which satisfies the syntactical rule
$$A_{j+1}(\boldsymbol{\varepsilon}) \supsetneqq A_j(\boldsymbol{\varepsilon}) \text{ or } A_{j+1}(\boldsymbol{\varepsilon}) = \emptyset, \qquad j \geq 0, \tag{5}$$

where $A_j(\boldsymbol{\varepsilon}) = \{1 \le k \le d : \varepsilon_j^{(k)} \neq 0\}$. It is an easy consequence of (5) that

$$\text{among } d+1 \text{ consecutive columns of a simple joint sparse form, there is at least one } \mathbf{0}. \tag{6}$$

Furthermore, the joint Hamming weight of this simple joint sparse form is minimum amongst all joint expansions of $x^{(1)}, \dots, x^{(d)}$. We also gave an algorithm to compute the simple joint sparse form of $x^{(1)}, \dots, x^{(d)}$. While our aim is to present a left-to-right algorithm for computing a minimal joint expansion, we discuss a few properties of the right-to-left algorithm and the simple joint sparse form since this will be useful for proving optimality of our algorithms in Sections 4 and 5.

The algorithm in [3] is formulated in terms of the integers $x^{(1)}, \dots, x^{(d)}$ and can easily be reformulated as an algorithm on the digits working from right to left, which we reproduce here as Algorithm 3. For a joint expansion $\boldsymbol{\eta}$, we will denote the output of Algorithm 3 when reading $\boldsymbol{\eta}$ by $\mathsf{algSJSF}(\boldsymbol{\eta})$.

The construction of our left-to-right algorithm in Section 4 will involve the use of Algorithm 3 for small blocks. Therefore, we will analyze its behaviour when its input consists of alternating greedy expansions.

Lemma 3. *Let $\boldsymbol{\eta}$ be a joint expansion of d integers such that $(\eta_j^{(k)})_{j\ge 0}$ satisfies* (1) *for $1 \le k \le d$ and let $\boldsymbol{\varepsilon} = \mathsf{algSJSF}(\boldsymbol{\eta})$. Then the following assertions hold.*

(1) *At all times, we have $|\varepsilon_j^{(k)}| \le 1$ for all j and k in Algorithm 3, which implies that the first inner loop is always empty.*
(2) *Let $\boldsymbol{\varepsilon}_j = \mathbf{0}$ for some $j \ge 0$. Then $(\boldsymbol{\varepsilon}_i)_{i>j} = \mathsf{algSJSF}((\boldsymbol{\eta}_i)_{i>j})$, i.e., if the algorithm outputs a zero column, the computation restarts completely.*
(3) *Let $\boldsymbol{\eta}_i = \mathbf{0}$ for all $i > j$ for some $j \ge 0$. Then $\boldsymbol{\varepsilon}_i = \mathbf{0}$ for all $i > j$, i.e., the simple joint sparse form of d integers is not longer than the longest alternating greedy expansion of one of the integers.*
(4) *Let $\ell := \max\{j \ge 0 : \boldsymbol{\eta}_j \neq \mathbf{0}\}$ and assume that there is a $1 \le k \le d$ such that $\eta_i^{(k)} = 0$ for all $i < \ell$. Let $(\boldsymbol{\varepsilon}'_{\ell-1}, \dots, \boldsymbol{\varepsilon}'_0) = \mathsf{algSJSF}((\boldsymbol{\eta}_{\ell-1}, \dots, \boldsymbol{\eta}_0))$, then $(\boldsymbol{\eta}_\ell, \boldsymbol{\varepsilon}'_{\ell-1}, \dots, \boldsymbol{\varepsilon}'_0)$ is an optimal joint expansion.*

Proof. (1) A digit of absolute value at least two can only be generated by the step $\varepsilon_{j+1}^{(k)} \leftarrow \varepsilon_{j+1}^{(k)} + \varepsilon_j^{(k)}$, since the first inner loop has not been executed by induction.

Assume that j is minimum such that such an assignment yields a number of absolute value at least 2. Just before this critical assignment, we must have $\varepsilon_{j+1}^{(k)} = \eta_{j+1}^{(k)} \neq 0$. Choose $i \le j$ maximal such that $\eta_i^{(k)} \neq 0$ and $\eta_\ell^{(k)} = 0$ for $i < \ell \le j$. By (1), we have $\eta_i^{(k)} = -\eta_{j+1}^{(k)}$. For $i \le \ell \le j$ we get $\varepsilon_\ell^{(k)} \in \{-\eta_{j+1}^{(k)}, 0\}$. This implies $\varepsilon_{j+1}^{(k)} + \varepsilon_j^{(k)} \in \{0, \eta_{j+1}^{(k)}\}$.

(2) If $\boldsymbol{\varepsilon}_j = \mathbf{0}$, we have $A_j = \emptyset$. This implies that in this step, none of the inner loops is nonempty, whence $\boldsymbol{\varepsilon}_{j+1} = \boldsymbol{\eta}_{j+1}$, i.e. the algorithm restarts.

(3) If $\boldsymbol{\eta}_{j+1} = \mathbf{0}$, we have $A_{j+1} = \emptyset$ and therefore $A_{j+1} \subseteq A_j$ is fulfilled, but the corresponding inner loop is empty. Therefore, $\boldsymbol{\varepsilon}_{j+1} = 0$ after step j. In the next

Algorithm 3 algSJSF: Simple Joint Sparse Form from Right to Left

Input: $(\eta_j^{(k)})_{\substack{1\le k\le d\\ 0\le j\le J}}$ arbitrary joint expansion of the integers $x^{(1)}, \dots, x^{(d)}$

Output: $(\varepsilon_j^{(k)})_{\substack{1\le k\le d\\ 0\le j\le J}}$ simple joint sparse form of $x^{(1)}, \dots, x^{(d)}$

$j \leftarrow 0$
$\boldsymbol{\eta}_{J+1} \leftarrow \mathbf{0}$
$\boldsymbol{\varepsilon}_0 \leftarrow \boldsymbol{\eta}_0$
$A_0 \leftarrow \{1 \le k \le d : \varepsilon_0^{(k)} \text{ is odd}\}$
for $j = 0$ to J **do**
  $\boldsymbol{\varepsilon}_{j+1} \leftarrow \boldsymbol{\eta}_{j+1}$
  for all k with $|\varepsilon_j^{(k)}| = 2$ **do**
    $\varepsilon_{j+1}^{(k)} \leftarrow \varepsilon_{j+1}^{(k)} + \varepsilon_j^{(k)}/2$
    $\varepsilon_j^{(k)} \leftarrow 0$
  end for
  $A_{j+1} \leftarrow \{1 \le k \le d : \varepsilon_{j+1}^{(k)} \text{ is odd}\}$
  if $A_{j+1} \subseteq A_j$ **then**
    for all $k \in A_{j+1}$ **do**
      $\varepsilon_{j+1}^{(k)} \leftarrow \varepsilon_{j+1}^{(k)} + \varepsilon_j^{(k)}$
      $\varepsilon_j^{(k)} \leftarrow -\varepsilon_j^{(k)}$
    end for
    $A_{j+1} \leftarrow \emptyset$
  else
    for all $k \in A_j \setminus A_{j+1}$ **do**
      $\varepsilon_{j+1}^{(k)} \leftarrow \varepsilon_{j+1}^{(k)} + \varepsilon_j^{(k)}$
      $\varepsilon_j^{(k)} \leftarrow -\varepsilon_j^{(k)}$
    end for
    $A_{j+1} \leftarrow A_j \cup A_{j+1}$
  end if
end for

step, $\boldsymbol{\varepsilon}_{j+1} = \boldsymbol{\varepsilon}_{j+2} = \mathbf{0}$, so the algorithm only produces $\mathbf{0}$ in all subsequent steps. Note that we used the fact that the first inner loop is empty.

(4) By Lemma 3 (3), the length of $\boldsymbol{\varepsilon}'$ is appropriate. We always have $k \notin A_i$ for $i < \ell$ when computing $\mathsf{algSJSF}((\boldsymbol{\eta}_\ell, \boldsymbol{\varepsilon}'_{\ell-1}, \dots, \boldsymbol{\varepsilon}'_0))$. Therefore, $k \in A_\ell \setminus A_{\ell-1}$ which implies that algSJSF would change column $\eta_\ell^{(m)}$ for $m \in A_{\ell-1} \setminus A_\ell$, but at this step, no zero column can be produced, so $(\boldsymbol{\eta}_\ell, \boldsymbol{\varepsilon}'_{\ell-1}, \dots, \boldsymbol{\varepsilon}'_0)$ is optimal, too.

□

Now we can state the crucial proposition.

Proposition 4. *For $1 \leq k \leq d$, let $(\eta_j^{(k)})_{0\leq j\leq J}$ be the alternating greedy expansion of some integer $x^{(k)}$. Assume that the set*

$$\{0 \leq i \leq J : \mathsf{algSJSF}((\boldsymbol{\eta}_j)_{i\leq j\leq J}) \text{ contains a column } \mathbf{0}\}$$

is nonempty and denote its maximum by I.

Write $\mathsf{algSJSF}((\boldsymbol{\eta}_j)_{0\leq j<I}) = (\boldsymbol{\varepsilon}'_j)_{0\leq j<I}$ and $\mathsf{algSJSF}((\boldsymbol{\eta}_j)_{I\leq j\leq J}) = (\boldsymbol{\varepsilon}''_j)_{I\leq j\leq J}$ and denote their concatenation by $\boldsymbol{\varepsilon} \in \{-1,0,1\}^{d\times J}$, i.e.,

$$\boldsymbol{\varepsilon}_j = \begin{cases} \boldsymbol{\varepsilon}'_j, & \text{if } j < I, \\ \boldsymbol{\varepsilon}''_j, & \text{if } j \geq I. \end{cases}$$

Then $\boldsymbol{\varepsilon}$ is a joint expansion of $x^{(1)}, \ldots, x^{(d)}$ of minimal joint Hamming weight.

Proof. First we note that by Lemma 3 (3), $(\boldsymbol{\varepsilon}'_j)_{0\leq j<I}$ and $(\boldsymbol{\varepsilon}''_j)_{I\leq j\leq J}$ are joint expansions of the same integers as $(\boldsymbol{\eta}'_j)_{0\leq j<I}$ and $(\boldsymbol{\eta}''_j)_{I\leq j\leq J}$, respectively. Therefore $\boldsymbol{\varepsilon}$ is a joint expansion of $x^{(1)}, \ldots, x^{(d)}$.

If ${\varepsilon'}_{I-1}^{(k)}$ is nonzero for some $1 \leq k \leq d$, say ${\varepsilon'}_{I-1}^{(k)} = 1$, then the integer $\sum_{j=0}^{I-1} {\varepsilon'}_j^{(k)} 2^j = \sum_{j=0}^{I-1} \eta_j^{(k)} 2^j$ is positive, too. Therefore the most significant nonzero digit of $(\eta_{I-1}^{(k)} \cdots \eta_0^{(k)})$ equals $+1$. This implies that the least significant nonzero digit of $(\eta_J^{(k)} \cdots \eta_I^{(k)})$ equals -1 by (1). We conclude that $(\eta_J^{(k)} \cdots \eta_I^{(k)} {\varepsilon'}_{I-1}^{(k)})$ satisfies (1).

Writing $\mathsf{algSJSF}((\boldsymbol{\eta}_J \cdots \boldsymbol{\eta}_I \boldsymbol{\varepsilon}'_{I-1})) = (\boldsymbol{\varepsilon}'''_j)_{I-1\leq j\leq J}$, we easily check that

$$\mathsf{algSJSF}(\boldsymbol{\eta}) = (\boldsymbol{\varepsilon}'''_J \cdots \boldsymbol{\varepsilon}'''_I \boldsymbol{\varepsilon}'''_{I-1} \boldsymbol{\varepsilon}'_{I-2} \cdots \boldsymbol{\varepsilon}'_0),$$

since Algorithm 3 uses a look-ahead of 1.

If $\boldsymbol{\varepsilon}'_{I-1} = \mathbf{0}$, we have $\mathsf{algSJSF}(\boldsymbol{\eta}) = \boldsymbol{\varepsilon}$ by Lemma 3 (2), and there is nothing to prove.

We now consider the case $\boldsymbol{\varepsilon}'_{I-1} \neq \mathbf{0}$. We note that $(\boldsymbol{\varepsilon}''_J \cdots \boldsymbol{\varepsilon}''_I \boldsymbol{\varepsilon}'_{I-1})$ is a joint expansion which contains at least one column $\mathbf{0}$ by definition of I. This implies that $(\boldsymbol{\varepsilon}'''_J \cdots \boldsymbol{\varepsilon}'''_{I-1})$ also contains a column $\mathbf{0}$, say $\boldsymbol{\varepsilon}'''_i = \mathbf{0}$ for some $i \geq I$. By Lemma 3 (2), we have $(\boldsymbol{\varepsilon}'''_J \cdots \boldsymbol{\varepsilon}'''_{i+1}) = \mathsf{algSJSF}((\boldsymbol{\eta}_J \cdots \boldsymbol{\eta}_{i+1}))$ and by definition of I, we conclude that there is no column $\mathbf{0}$ in $(\boldsymbol{\varepsilon}'''_J \cdots \boldsymbol{\varepsilon}'''_{i+1})$. Therefore, we see that $(\boldsymbol{\varepsilon}''_J \cdots \boldsymbol{\varepsilon}''_I \boldsymbol{\varepsilon}'_{I-1})$ and $(\boldsymbol{\varepsilon}'''_J \cdots \boldsymbol{\varepsilon}'''_{I-1})$ have the same joint Hamming weight, whence $\boldsymbol{\varepsilon}$ and $\mathsf{algSJSF}(\boldsymbol{\eta})$ have the same joint Hamming weight, too. ☐

4. Computing a Minimal Joint Expansion from Left to Right

It is the aim of this section to derive an algorithm (which can be realized by a transducer) transforming the alternating greedy expansions of $x^{(1)}, \ldots, x^{(d)}$ (seen as a joint expansion) to an expansion of the same (and therefore optimal) joint Hamming weight as the simple joint sparse form. We note that it is impossible to compute the simple joint sparse form from left to right (the fractal structures discussed in [3] prove this; cf. also the example for $d = 1$ in [5]), we can only compute a joint expansion of the same Hamming weight.

Proposition 4 allows us to split the computation of a minimal joint expansion into several pieces. From (6) we also see that $J \geq I \geq J - d$ (if $J \geq d$). Therefore, we can use Algorithm 4 to compute a minimal joint expansion from left to right.

Algorithm 4 Computing a Minimal Joint Expansion from Left to Right Using algSJSF

Input: $x^{(1)}, \ldots, x^{(d)}$ integers
Output: joint expansion $\boldsymbol{\varepsilon}$ of $x^{(1)}, \ldots, x^{(d)}$ of minimal joint Hamming weight.
$(\eta_j^{(k)})_{0\le j\le J} \leftarrow$ Alternating greedy expansion of $x^{(k)}$, $1 \le k \le d$
while $J \ge 0$ **do**
$I \leftarrow \max(\{\max(J-d,0) \le i \le J : \mathsf{algSJSF}((\boldsymbol{\eta}_j)_{i\le j\le J})$ contains a column $\mathbf{0}\} \cup \{0\})$
$(\boldsymbol{\varepsilon}_j)_{I\le j\le J} \leftarrow \mathsf{algSJSF}((\boldsymbol{\eta}_j)_{I\le j\le J})$
$J \leftarrow I-1$
end while

We note that for fixed d, Algorithm 4 can be implemented as a transducer automaton transforming the unsigned binary expansions to a minimal joint expansion, where "unsigned" means the unique expansion with digits $\{0,-1\}$ if the represented number is negative. In this case, we have to modify the transducer in Figure 1 in the appropriate way.

For $d = 2$, such a transducer automaton has been explicitly constructed in [4] (we note that in one case, we introduced a small variation which does not change the Hamming weight).

We summarize our findings as follows.

Theorem 5. *Let $d \ge 1$. There is a transducer automaton to transform the unsigned joint expansion of integers $x^{(1)}, \ldots, x^{(d)}$ to a joint expansion of minimal joint Hamming weight. It can be realized by combining the transducer in Figure 1 and Algorithm 4.*

5. Direct Algorithm without Preprocessing

In this section we introduce another left-to-right algorithm that computes a minimal joint signed binary expansion of d integers from their unsigned binary expansions. This approach is advantageous in that it does not need any preprocessing and works based on simple column scanning and bitwise replacement. Furthermore, it makes less replacements and does not use algSJSF as an intermediate step.

In Algorithm 5, we scan the unsigned binary expansions of the d integers from the most significant bit column $(\ell-1)$, towards the least significant bit column (0), $d+1$ columns at a time.

Algorithm 5 consists of two steps:

Step 1: Converting the unsigned binary input to the alternating greedy expansions.
Step 2: Making replacements on the alternating greedy expansions.

In Step 2, three conditions must be satisfied before a replacement takes place. These three conditions are:

C1: *LeftmostIsNonzero* $\neq \emptyset$
C2: For each $k \in$ *LeftmostIsNonzero* there is an i with

$$j > i \ge \mathit{EndComputingAlternatingGreedy}$$

Algorithm 5 Computing a Minimal Joint Expansion from the Unsigned Binary Expansion from Left to Right

Input: ℓ-bit unsigned binary expansion $(\eta^{(k)}_{\ell-1}, \eta^{(k)}_{\ell-2}, \dots, \eta^{(k)}_1, \eta^{(k)}_0)$ of d $(d \geq 1)$ integers $x^{(k)}$ $(1 \leq k \leq d)$.

Output: $(\ell+1)$-bit signed binary expansion $(\varepsilon^{(k)}_{\ell}, \varepsilon^{(k)}_{\ell-1}, \dots, \varepsilon^{(k)}_1, \varepsilon^{(k)}_0)$ of $x^{(k)}$ $(1 \leq k \leq d)$ such that the joint Hamming weight is minimum.

$\eta^{(k)}_{\ell} \leftarrow 0$ for each k with $1 \leq k \leq d$
$\eta^{(k)}_{-1} \leftarrow 0$ for each k with $1 \leq k \leq d$
StartComputingAlternatingGreedy $\leftarrow \ell$
$j \leftarrow \ell$
while $j \geq 0$ **do**
  EndComputingAlternatingGreedy $\leftarrow \max(j-d, 0)$
  for $1 \leq k \leq d$ **do**
    for *StartComputingAlternatingGreedy* $\geq i \geq$ *EndComputingAlternatingGreedy* **do**
      $\varepsilon^{(k)}_i \leftarrow \eta^{(k)}_{i-1} - \eta^{(k)}_i$
    end for
  end for
  StartComputingAlternatingGreedy $\leftarrow$ *EndComputingAlternatingGreedy* $- 1$
  LeftmostIsNonzero $\leftarrow \{1 \leq k \leq d : \varepsilon^{(k)}_j \neq 0\}$
  if *LeftmostIsNonzero* $\neq \emptyset$ and for each $k \in$ *LeftmostIsNonzero* there is an i with $j > i \geq$ *EndComputingAlternatingGreedy* satisfying $\varepsilon^{(k)}_i \neq 0$ **then**
    NextNonzeroLocation$[k]$ $\leftarrow \max\{i : j > i$ and $\varepsilon^{(k)}_i \neq 0\}$ for each $k \in$ *LeftmostIsNonzero*
    MinNextNonzeroLocation $\leftarrow \min\{$*NextNonzeroLocation*$[k]$: $k \in$ *LeftmostIsNonzero*$\}$
    RightmostNonzeroLocation$[k]$ $\leftarrow \min\{j > i \geq$ *MinNextNonzeroLocation* $: \varepsilon^{(k)}_i \neq 0\}$ for $1 \leq k \leq d$
    if $\{i : j > i \geq$ *MinNextNonzeroLocation*$\} = \{$*RightmostNonzeroLocation*$[k] : 1 \leq k \leq d\}$ **then**
      for all $k \in$ *LeftmostIsNonzero* **do**
        $\varepsilon^{(k)}_i \leftarrow \varepsilon^{(k)}_j$ for each i with $j-1 \geq i \geq$ *NextNonzeroLocation*$[k]$
        $\varepsilon^{(k)}_j \leftarrow 0$
      end for
      $j \leftarrow$ *MinNextNonzeroLocation* $- 1$
    else
      $j \leftarrow j - 1$
    end if
  else
    $j \leftarrow j - 1$
  end if
end while

satisfying $\varepsilon_i^{(k)} \neq 0$.

C3: $\{i : j > i \geq \mathit{MinNextNonzeroLocation}\} = \{\mathit{RightmostNonzeroLocation}[k] : 1 \leq k \leq d\}$

If all three conditions are satisfied then the leftmost column of the $d+1$ columns being scanned will be converted from nonzero to zero. The policy is to replace $x0\ldots 0\overline{x}$ by $0x\ldots xx$ ($x \in \{-1, 1\}$) in each row k with $k \in \mathit{LeftmostIsNonzero}$. Algorithm 5 then skips the columns involved in the replacement and restarts the scanning.

If one or more of the three conditions are not satisfied then Algorithm 5 moves rightwards by one column and restarts the scanning.

Properties of Algorithm 5 are stated in the following lemmas and theorems.

Lemma 6. *Let $x^{(1)}, \ldots, x^{(d)}$ be integers with alternating greedy expansions $(\varepsilon_i^{(k)})_{1\leq k\leq d}$ and let J be an integer such that $j = J$ holds at some stage of the execution of Algorithm 5 (running on the unsigned expansions of $x^{(1)}, \ldots, x^{(d)}$). We set*

$$I := \max(\{0 \leq i \leq J : \mathsf{algSJSF}((\boldsymbol{\varepsilon}_J, \ldots, \boldsymbol{\varepsilon}_i)) \text{ contains a } \mathbf{0}\} \cup \{0\}).$$

Then the output columns $(\boldsymbol{\varepsilon}'_J, \ldots, \boldsymbol{\varepsilon}'_I)$ of Algorithm 5 have the same joint Hamming weight as $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_J, \ldots, \boldsymbol{\varepsilon}_I))$.

Proof. We prove the lemma by induction on J. We note that $I \geq J - d$ by (6).

If **C1** is violated, then we clearly have $I = J$ and $\boldsymbol{\varepsilon}'_J = \mathbf{0}$, which is best possible. So we can assume that **C1** holds. If **C2** is not satisfied, Lemma 3 (4) shows that column $\boldsymbol{\varepsilon}_J$ can be left unchanged. Decreasing J to $J-1$ does not alter I, so $(\boldsymbol{\varepsilon}'_{J-1}, \ldots, \boldsymbol{\varepsilon}'_I)$ has the same joint Hamming weight as $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_{J-1}, \ldots, \boldsymbol{\varepsilon}_I))$ by induction. Thus $(\boldsymbol{\varepsilon}'_J, \ldots, \boldsymbol{\varepsilon}'_I)$ has the same joint Hamming weight as $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_J, \ldots, \boldsymbol{\varepsilon}_I))$ by Lemma 3 (4).

Therefore, we can assume that **C1** and **C2** hold. We assume that **C3** is violated. We choose the smallest i with

$$J > i \geq \mathit{MinNextNonzeroLocation} \text{ and } i \notin \{\mathit{RightmostNonzeroLocation}[k] : 1 \leq k \leq d\}.$$

In this case it is easily seen that $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_i, \ldots, \boldsymbol{\varepsilon}_{\mathit{MinNextNonzeroLocation}})) = (\mathbf{0}, \mathbf{?}, \ldots, \mathbf{?})$, where $\mathbf{?}$ stands for some uninteresting digit vector. We get $I \geq \mathit{MinNextNonzeroLocation}$. In the case $I > \mathit{MinNextNonzeroLocation}$, there is a $k \in \mathit{LeftmostIsNonzero}$ satisfying $\mathit{NextNonzeroLocation}[k] < I$. By Lemma 3 (4) this means that we can keep $\boldsymbol{\varepsilon}_J$ and decrease J to $J-1$ without changing I. Therefore, we are left with the case $I = \mathit{MinNextNonzeroLocation}$. Lemma 3 (2) says that $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_J, \ldots, \boldsymbol{\varepsilon}_i, \ldots, \boldsymbol{\varepsilon}_I))$ restarts after dealing with column i. By maximality of I, the output of $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_J, \ldots, \boldsymbol{\varepsilon}_i))$ does not contain a $\mathbf{0}$. Therefore, once again, we can decrease J to $J-1$ without changing I and use the induction hypothesis.

Finally, we consider the case that all three conditions **C1**, **C2**, and **C3** are satisfied. Then $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_J, \ldots, \boldsymbol{\varepsilon}_{\mathit{MinNextNonzeroLocation}})) = (\mathbf{0}, \mathbf{?}, \ldots, \mathbf{?})$, where all $\mathbf{?} \neq \mathbf{0}$. This implies that $I \geq \mathit{MinNextNonzeroLocation}$. If $I > \mathit{MinNextNonzeroLocation}$, there is a $k \in \mathit{LeftmostIsNonzero}$ with $\mathit{NextNonzeroLocation}[k] < I$, hence $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_J, \ldots, \boldsymbol{\varepsilon}_I))$ cannot contain a zero column, contradiction. Therefore we have $I = \mathit{MinNextNonzeroLocation}$, and it is clear that $(\boldsymbol{\varepsilon}'_J, \dots, \boldsymbol{\varepsilon}'_I)$ and $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_J, \dots, \boldsymbol{\varepsilon}_I))$ both have exactly one zero column.

□

Theorem 7. *The output of Algorithm 5 has minimal joint Hamming weight among any signed binary expansions of the d given integers.*

Proof. This is a direct consequence of Proposition 4 and Lemma 6. □

Remark 8. Since the output of Algorithm 5 has minimal joint Hamming weight, the results in [4, Table 3] apply for our algorithm, too: If $H_{\ell,d}$ denotes the joint Hamming weight of a minimal joint expansion of d random ℓ-digit integers, the asymptotic expected value $\mathbb{E}(H_{\ell,d})$ and the asymptotic variance $\mathbb{V}(H_{\ell,d})$ can be found in Table 1. By Hwang's [6] quasi-power theorem, we also know a central limit theorem, cf. [4].

d	$\frac{1}{\ell}\mathbb{E}(H_{\ell,d})$	$\frac{1}{\ell}\mathbb{V}(H_{\ell,d})$
1	$\frac{1}{3}$	$\frac{2}{27}$
2	$\frac{1}{2}$	$\frac{1}{16}$
3	$\frac{23}{39}$	$\frac{2800}{59319}$
4	$\frac{115}{179}$	$\frac{210368}{5735339}$
5	$\frac{4279}{6327}$	$\frac{7565047808}{253275687783}$
6	$\frac{152821}{218357}$	$\frac{263523314106368}{10411213601145293}$
7	$\frac{21292819}{29681427}$	$\frac{5775339222194344967040}{2614895455649204000001483}$

TABLE 1. Asymptotic means and variances of the minimal joint Hamming weight of d random ℓ-digit integers.

Remark 9. We remark that the Algorithms 4 and 5 both produce a joint expansion of minimal joint Hamming weight, but they do not necessarily produce the same output. However, the positions of the **0**-columns can be shown to be the same. As an example, consider the input (already converted to its alternating greedy expansion)

$$\begin{pmatrix} 0 & 1 & 0 & \bar{1} & 0 & 1 & 0 & \bar{1} \\ 1 & 0 & 0 & \bar{1} & 1 & 0 & 0 & \bar{1} \end{pmatrix}.$$

Algorithm 4 produces the output

$$\begin{pmatrix} 1 & \bar{1} & 0 & 0 & \bar{1} & \bar{1} & 0 & \bar{1} \\ 1 & 0 & 0 & 0 & \bar{1} & 0 & 0 & \bar{1} \end{pmatrix},$$

whereas Algorithm 5 produces

$$\begin{pmatrix} 0 & 1 & 0 & 0 & \bar{1} & \bar{1} & 0 & \bar{1} \\ 1 & 0 & 0 & 0 & \bar{1} & 0 & 0 & \bar{1} \end{pmatrix},$$

since changing the first column does not alter the joint Hamming weight (cf. Lemma 3 (4)).

Lemma 10. *Let $J \geq d$ be the index of a column such that we have $j = J$ at some stage of Algorithm 5. Then at least one of the columns $J, \ldots, J-d$ will be a zero column in the output of Algorithm 5.*

Proof. Since at least one of $d+1$ consecutive output columns of $\mathsf{algSJSF}$ is zero by (6), we have $I \geq J-d$ for the I defined in Lemma 6. By the same Lemma 6, at least one column (in the same range) of the output of Algorithm 5 must also be a **0**. □

Theorem 11. *Among $2d+1$ consecutive columns of the output of Algorithm 5, there is at least one* **0**.

Proof. By Lemma 10, in the worst case, Algorithm 5 makes the leftmost column out of $d+1$ consecutive nonzero columns a zero column. Then the remaining d nonzero columns that have been replaced are skipped. If we are unlucky enough then the next d columns that will be looked at might be all nonzero and impossible to be replaced at all. Thus in the worst case only one zero column results out of $2d+1$ consecutive columns. □

When implemented in hardware, Algorithm 5 leads to a significant reduction in hardware overhead. This is because the binary input $\eta_i^{(k)}$ is never used again after the calculation of $\varepsilon_i^{(k)}$. Therefore the input array $\eta_j^{(k)}$ and the output array $\varepsilon_j^{(k)}$ can share the same memory space.

During the computation, the number of active columns, i.e., columns that are being scanned, is at most $d+1$. If the output of Algorithm 5 is input to a realtime processor for further operation, then the amount of required memory could be reduced to as low as $d \times (d+1)$ signed binary bits.

6. w-NAF

Let $w \geq 1$ be an integer. The w-NAF of an integer n is the unique binary expansion with digits in $\mathcal{D}_w := \{0, \pm 1, \pm 3, \pm 5, \ldots, \pm(2^w - 3), \pm(2^w - 1)\}$ such that any two nonzero digits are separated by at least w zeros. In the case $w = 1$, the 1-NAF is usually just called NAF, cf. Reitwiesner [10]. It is clear that the w-NAF can be computed from right to left by selecting the rightmost digit according to $n \bmod 2^{w+1}$.

Avanzi [1] showed that the w-NAF has minimal Hamming weight amongst all signed binary expansions with digits of absolute value less than 2^w. He also gave an algorithm for computing a $\mathcal{D}_w$-expansion of minimal Hamming weight from left to right. In this section, we show that this can be accomplished in a very natural way by using the alternating greedy expansion.

Algorithm 6 Computing a Minimal Binary Expansion with Digits of Absolute Value at most $2^w - 1$ from Left to Right

Input: Alternating greedy expansion $(\eta_J, \ldots, \eta_0)$ of n; $w \geq 1$
Output: $\mathcal{D}_w$-expansion $\boldsymbol{\varepsilon}$ of n of minimal Hamming weight

```
ε ← 0
j ← J
while j ≥ 0 do
  if η_j = 0 then
    j ← j − 1
  else
    t ← max(j − w, 0)
    while η_t = 0 do
      t ← t + 1
    end while
    ε_t ← Σ_{ℓ=t}^{j} η_ℓ 2^{ℓ−t}
    j ← j − w − 1
  end if
end while
```

Theorem 12. *Algorithm 6 computes a binary $\mathcal{D}_w$-expansion of n whose Hamming weight is minimum amongst all binary expansions of n with digits $\{-(2^w - 1), \ldots, (2^w - 1)\}$.*

We remark that we compute a binary expansions whose nonzero digits are odd; its Hamming weight is still minimum amongst all binary expansions which also include even digits.

Proof. Since the algorithm works by replacing $(\eta_j \cdots \eta_t)$ by $(0 \cdots 0 \sum_{\ell=t}^{j} \eta_\ell 2^{\ell-t})$, we certainly have $\mathsf{value}(\boldsymbol{\varepsilon}) = \mathsf{value}(\boldsymbol{\eta})$. Furthermore we have $|\varepsilon_t| \leq 2^{j-t} \leq 2^w$ by (3) and (4). Since $\eta_t \neq 0$, we conclude that all $\varepsilon_t \in \mathcal{D}_w$.

To prove optimality, we use induction on J. Of course, we may assume that $\eta_J \neq 0$. We consider the first replacement $\varepsilon_t \leftarrow \sum_{\ell=t}^{J} \eta_\ell 2^{\ell-t}$ and set $m = n - \varepsilon_t 2^t$. It is clear that m has (possibly unbalanced) alternating greedy expansion $(\eta_{J-w-1}, \ldots, \eta_0)$. We define h to be the Hamming weight of the w-NAF of m which, by induction, equals the Hamming weight of the output of Algorithm 6 with input $(\eta_{J-w-1}, \ldots, \eta_0)$. The $\boldsymbol{\varepsilon}$ produced by Algorithm 6 with input $(\eta_J, \ldots, \eta_0)$ has therefore Hamming weight $h+1$.

We write the w-NAFs of m and n as

$$m = \sum_{k=1}^{h} a_k 2^{r_k}, \qquad n = \sum_{k=1}^{h'} b_k 2^{s_k},$$

respectively, where $a_k \neq 0$ and $b_k \neq 0$ for all k. We have to prove that $h' \geq h+1$.

The w-NAF condition implies that $r_{k+1} \geq r_k + w + 1$ for $k \geq 1$. Using (3) and (4), we have

$$\begin{aligned} 2^{r_h} \leq |a_h 2^{r_h}| &= \left| m - \sum_{k=1}^{h-1} a_k 2^{r_k} \right| \leq 2^{J-w-1} + (2^w - 1) \sum_{k=1}^{h-1} 2^{r_k} \\ &= 2^{J-w-1} + \sum_{k=1}^{h-1} 2^{r_k + w} - \sum_{k=1}^{h-1} 2^{r_k} \leq 2^{J-w-1} + \sum_{k=2}^{h} 2^{r_k - 1} - \sum_{k=1}^{h-1} 2^{r_k} \\ &\leq 2^{J-w-1} + 2^{r_h - 1} - 2^{r_1} < 2^{J-w-1} + 2^{r_h - 1}, \end{aligned}$$

which implies $r_h - 1 < J - w - 1$ and therefore $r_h \leq J - w - 1$. We also conclude that $|a_h| \leq 2^{J-w-1-r_h}$.

Furthermore, for $k \leq h-1$, the quantities a_k and r_k only depend on $m \bmod 2^{r_{h-1}+w+1}$. Since $r_{h-1} + w + 1 \leq r_h \leq J - w - 1$ and $m \equiv n \pmod{2^{J-w}}$, we conclude that $a_k = b_k$ and $r_k = s_k$ for $1 \leq k \leq h-1$. We obtain $a_h + \varepsilon_t 2^{t-r_h} = \sum_{k=h}^{h'} b_k 2^{s_k - r_h}$. Since $t \geq J - w > r_h$ and a_h is odd, we get $s_h = r_h$.

By (3) and (4) we have

$$\left| a_h + \varepsilon_t 2^{t-r_h} \right| \geq 2^{J-t-1} \cdot 2^{t-r_h} - 2^{J-w-1-r_h} = 2^{J-r_h-1} - 2^{J-w-1-r_h} \geq 2^w - 1. \tag{7}$$

If equality holds, $|\varepsilon_t| = 2^{J-t-1}$, which implies ($\varepsilon_t$ is odd) that $t = J-1$, $\operatorname{sign}(a_h) = \operatorname{sign}(m) = -\operatorname{sign}(\eta_t) = \operatorname{sign}(\varepsilon_t)$ and therefore $|a_h + \varepsilon_t 2^{t-r_h}| = |a_h| + |\varepsilon_t 2^{t-r_h}|$. This implies that equality cannot hold in (7). We conclude that $h' > h$. $\square$

References

[1] R. M. Avanzi, *A note on the sliding window integer recoding and its left-to-right analogue*, to appear in Proc. SAC 2004, August 9–10, University of Waterloo, Waterloo, Ontario, Canada.

[2] P. Grabner, C. Heuberger, and H. Prodinger, *Subblock occurrences in signed digit representations*, Glasg. Math. J. **45** (2003), 427–440.

[3] 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.

[4] P. J. Grabner, C. Heuberger, H. Prodinger, and J. Thuswaldner, *Analysis of linear combination algorithms in cryptography*, Preprint, available at `http://www.opt.math.tu-graz.ac.at/~cheub/publications/Windows.pdf`.

[5] C. Heuberger, *Minimal expansions in redundant number systems: Fibonacci bases and greedy algorithms*, to appear in Period. Math. Hungar., available at `http://www.opt.math.tu-graz.ac.at/~cheub/publications/minredfibo.ps`.

[6] H.-K. Hwang, *On convergence rates in the central limit theorems for combinatorial structures*, European J. Combin. **19** (1998), 329–343. MR **99c:**60014

[7] M. Joye and S.-M. Yen, *Optimal left-to-right binary signed digit recoding*, IEEE Transactions on Computers **49** (2000), no. 7, 740–748.

[8] R. Katti and X. Ruan, *Left-to-right binary signed-digit recoding for elliptic curve cryptography*, Proc. 2004 IEEE International Symposium on Circuits and Systems.

[9] H. Prodinger, *On binary representations of integers with digits* $-1, 0, 1$, Integers **0** (2000), A08, available at `http://www.integers-ejcnt.org/vol0.html`.

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

[11] J. A. Solinas, *Low-weight binary representations for pairs of integers*, Preprint.

(C. Heuberger) Institut für Mathematik B, Technische Universität Graz, Steyrergasse 30, 8010 Graz, Austria
E-mail address: `clemens.heuberger@tugraz.at`

(R. Katti) Department of Electrical and Computer Engineering, North Dakota State University, Fargo, North Dakota 58105, U.S.A.
E-mail address: `rajendra.katti@ndsu.nodak.edu`

(H. Prodinger) The John Knopfmacher Centre for Applicable Analysis and Number Theory, School of Mathematics, University of the Witwatersrand, P. O. Wits, 2050 Johannesburg, South Africa
E-mail address: `helmut@maths.wits.ac.za`

(X. Ruan) Department of Electrical and Computer Engineering, North Dakota State University, Fargo, North Dakota 58105, U.S.A.
E-mail address: `xiaoyou.ruan@ndsu.nodak.edu`