THE ALTERNATING GREEDY EXPANSION AND APPLICATIONS TO COMPUTING DIGIT EXPANSIONS FROM LEFT-TO-RIGHT IN CRYPTOGRAPHY

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

Abstract. The central topic of this paper is the *alternating greedy expansion* of integers, which is defined to be a binary expansion with digits $\{0, \pm 1\}$ with the property that the nonzero digits have alternating signs. 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 also compute an expansion equivalent to the so-called w-NAF from left to right using the alternating greedy expansion.

1. Introduction

A signed binary expansion of an integer n is a radix 2 expansion of n where the digits belong to the set $\{0, \pm 1\}$. The central topic of this paper is the *(balanced) alternating greedy expansion*, i.e., a signed binary expansion of n with the property that two consecutive nonzero digits—even if separated by some digits 0—are opposite.

We use this expansion for constructing algorithms that compute optimal digital expansions—which are useful in cryptography—*from left to right*, i.e., from the most significant to the least significant digits. The usual double-and-add algorithms for computing scalar multiples (or linear combinations) of points in an Abelian group—for example the group of rational points on an elliptic curve—need the digits of a signed binary expansion of the scalar from left to right if precomputations are used. Therefore, this direction has the advantage that no extra storage for the digits is needed. Optimality means that the Hamming weight of the expansion—which essentially equals the number of group additions—is minimum. The relevant feature of the alternating greedy expansion is that it blocks carries in a certain sense, which makes a transformation from left to right more predictable.

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

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

[¶] These authors are supported in part by the grant CCR-0429523 of the United State National Science Foundation.

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

proof of uniqueness. We conclude that section by some estimates for alternating greedy expansions.

Sections 3 to 5 are devoted to the first application: Given integers $x^{(1)}, \dots, x^{(d)}$, we consider signed binary expansions of these integers written as rows of an array. The aim is to minimize the *joint Hamming weight* which is defined to be the number of nonzero columns. This equals the number of group additions when computing the linear combination $x^{(1)}P_1 + \cdots + x^{(d)}P_d$ of some points $P_1, \dots, P_d$ on the elliptic curve. We give two algorithms to compute such a minimal joint expansion from left to right. 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.

In Section 6 we consider expansions of a single integer with digits $\{-(2^w-1), -(2^w-3), \dots, -3, -1, 0, 1, 3, \dots, 2^w-3, 2^w-1\}$ of minimal Hamming weight. The corresponding (well-known) right to left method is known as the sliding window method, it yields the so-called w-NAF, i.e., the unique expansion with the same digits where all non-zero digits are separated by at least w zeros. We show that in this case also, a left-to-right algorithm can be easily obtained by first converting the input to its alternating greedy expansion.

Signed binary expansions have been used for a long time, cf. Booth [2]. We note that the alternating greedy expansion has been introduced in [5], where we gave 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, we also used this expansion for computing a minimal joint expansion of 2 integers from left to right. The minimality proof relied on a counting argument using generating functions, thus it was not very intuitive and cannot be used for general dimensions d. A similar left-to-right algorithm was also presented in [9]. Although we never stated it explicitly, the left-to-right algorithm for computing minimal signed binary expansions of single integers as discussed in Joye and Yen [8] and [6] can also be seen to be based on the alternating greedy expansion.

Left-to-right algorithms for minimal expansions with digits $\{-(2^w-1), -(2^w-3), \dots, -3, -1, 0, 1, 3, \dots, 2^w-3, 2^w-1\}$ have also been developed by Avanzi [1] and Muir and Stinson [11]. Our goal here is to show that the alternating greedy expansion is the "natural" way to get such an algorithm, since the underlying "meta-algorithm" is to apply the well-known right-to-left algorithm on blocks of alternating greedy expansions.

We will repeatedly use transducer automata to transform digit expansions, cf. for instance [10].

A final remark on terminology: in contrast to [5], the "greedy" character of the alternating greedy expansion does not play a dominant rôle in this paper. Nevertheless, we decided not to change the name of the expansion by omitting the word "greedy" in order to emphasize that we are indeed speaking on the same expansion as introduced in [5].

2. The Alternating Greedy Expansion

Throughout this paper, a *(signed binary) expansion of an integer* n is an $\boldsymbol{\varepsilon} = (\varepsilon_j)_{j\in\mathbb{N}_0} = (\ldots,\varepsilon_2,\varepsilon_1,\varepsilon_0) \in \{-1,0,1\}^{\mathbb{N}_0}$ 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.

The *unisigned binary expansion* of an integer n is the unique signed binary expansion of n with digits in $\{0,1\}$ or $\{0,-1\}$ only, depending on the sign of n.

We recall the definition of an *alternating greedy expansion* of integers as introduced in [5]. A signed binary expansion $\boldsymbol{\varepsilon}$ of an integer n is called a *(balanced) alternating greedy expansion of* n, if it satisfies the conditions

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

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

The existence of the alternating greedy expansion of an integer has been proved in [5] by a transducer automaton and we stated there that it is unique without giving a detailed proof. We will give a detailed proof below by exhibiting an (algorithmic) bijection between the unisigned binary expansions and the alternating greedy expansions preserving the value of the expansion.

Algorithm 1 $\mathsf{algBINtoAGE}$: Computing the Alternating Greedy Expansion from the Unisigned Binary Expansion from Left to Right

```
Input: Unisigned binary expansion $\boldsymbol{\eta} = (\eta_{J-1},\eta_{J-2},\ldots,\eta_1,\eta_0)$ of an integer $n$.
Output: Alternating greedy expansion $\boldsymbol{\varepsilon} = (\varepsilon_J,\varepsilon_{J-1},\ldots,\varepsilon_1,\varepsilon_0)$ of $n$.
  $\eta_J \leftarrow 0$
  $\eta_{-1} \leftarrow 0$
  for $J \geq j \geq 0$ do
    $\varepsilon_j \leftarrow \eta_{j-1} - \eta_j$
  end for
```

We first consider Algorithm 1 operating from left to right. A direct computation shows that $\mathsf{value}(\boldsymbol{\varepsilon}) = \mathsf{value}(\boldsymbol{\eta})$. We observe that 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 write $\bar{1}$ for the digit -1. The label of a state corresponds to the last input digit read, the superscript $+$ or $-$ remembers the sign of the last nonzero input digit read. If we restrict the transducer to the input alphabet $\{0,1,\perp\}$ only, we obtain the transducer (with other labels for the states) shown in Figure 2 of [5]. Considering the output of the transducer in Figure 1, it turns out that this output is indeed an alternating greedy expanion. Thus Algorithm 1 is correct. This also re-proves that every integer has an alternating greedy expansion.

Next, we consider Algorithm 2. We observe that it can be realized by the transducer in Figure 2. The labels of the states correspond to the next output digit which has to

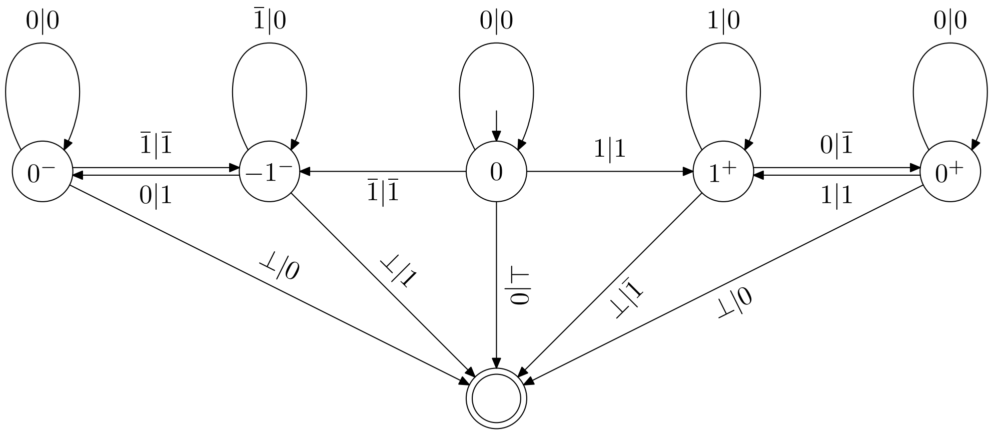

Figure 1. Transducer realizing algBINtoAGE from left to right. The symbol $\perp$ denotes the end of the sequence.

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

```
Input: Alternating greedy expansion $\boldsymbol{\varepsilon} = (\varepsilon_J, \varepsilon_{J-1}, \dots, \varepsilon_1, \varepsilon_0)$ of an integer $n$.
Output: The unisigned binary expansion $(\eta_{J-1}, \eta_{J-2}, \dots, \eta_1, \eta_0)$ of $n$.
  $\eta_J \leftarrow 0$
  for $j = J-1$ down to $0$ do
    $\eta_j \leftarrow \eta_{j+1} + \varepsilon_{j+1}$
  end for
```

be written (note that Algorithm 2 computes ε_j from the digits in position $(j+1)$), the superscript remembers the last nonzero input digit read. This shows that the output of Algorithm 2 is indeed a unisigned binary expansion. Furthermore, a direct computation shows that both compositions algBINtoAGE ∘ algAGEtoBIN and algAGEtoBIN ∘ algBINtoAGE are the identity map, thus Algorithms 1 and 2 are inverse to each other. This also shows that Algorithm 2 preserves the value of the expansion. We conclude that Algorithm 2 is correct, too. Moreover, this also shows that each integer n has exactly one alternating greedy expansion, hence we call it *the* alternating greedy expansion of n.

Since the loop in Algorithm 1 can be executed in parallel, we can compute the alternating greedy expansion of n by digitwise subtraction of the unisigned binary expansions of $2n$ and n. This gives rise to the explicit digit formula

$$\varepsilon_j = \eta_{j-1} - \eta_j = \operatorname{sign}(n)\left(\left\lfloor\frac{|n|}{2^{j-1}}\right\rfloor - 3\left\lfloor\frac{|n|}{2^{j}}\right\rfloor + 2\left\lfloor\frac{|n|}{2^{j+1}}\right\rfloor\right),$$

where η_j and ε_j denote the digit in position j of the unisigned binary and the alternating greedy expansion of n, respectively.

The right-to-left version of Algorithm 1 can be realized by the transducer in Figure 3.

We summarize our findings on the alternating greedy expansion in the following theorem.

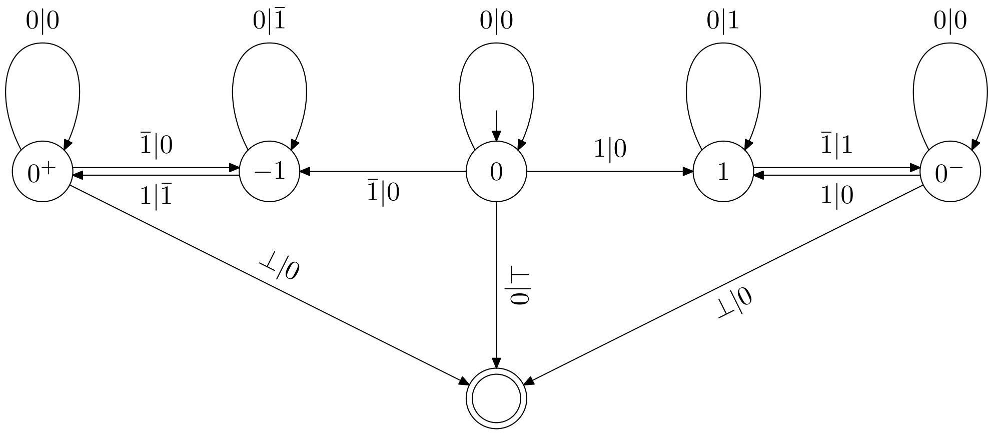

FIGURE 2. Transducer realizing algAGEtoBIN from left to right.

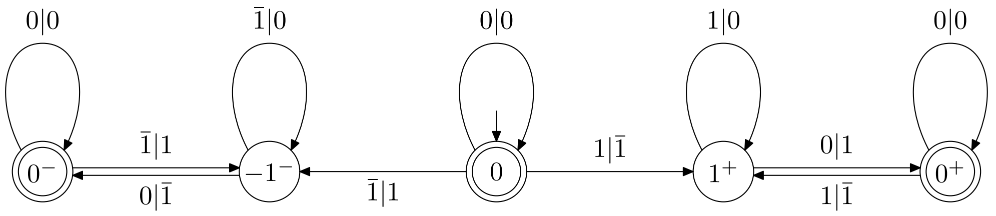

FIGURE 3. Transducer realizing algBINtoAGE from right to left.

Theorem 1. *For every integer n, there is a unique alternating greedy expansion $\boldsymbol{\varepsilon}$ of n. It can be computed by digitwise subtraction of the unisigned binary expansions of $2n$ and n, by the digit formula*

$$\varepsilon_j = \operatorname{sign}(n)\left(\left\lfloor\frac{|n|}{2^{j-1}}\right\rfloor - 3\left\lfloor\frac{|n|}{2^{j}}\right\rfloor + 2\left\lfloor\frac{|n|}{2^{j+1}}\right\rfloor\right), \tag{3}$$

by Algorithm 1, by the transducer in Figure 1 from left to right and by the transducer in Figure 3 from right to left. It can be converted back to the unisigned expansion of n by Algorithm 2, realized by the transducer in Figure 2 from left to right.

The digit formula (3) shows that the alternating greedy expansion is a member of the class of expansions which can be written as a digitwise linear combination of the unisigned binary expansion, cf. Prodinger [12]. Such representations enable one to perform a detailed analysis of the frequency of digits (or subblocks) as described in [3], for instance.

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

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

is called an *unbalanced alternating greedy expansion* of n.

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

(1) *We have* $\operatorname{sign}(n) = \operatorname{sign}(\varepsilon_J)$.

(2) *If $\boldsymbol{\varepsilon}$ is a balanced alternating greedy expansion, we have*

$$2^{J-1} \leq |n| < 2^J. \tag{4}$$

(3) *If $\boldsymbol{\varepsilon}$ is an unbalanced alternating greedy expansion, we have*

$$2^{J-1} < |n| \leq 2^J. \tag{5}$$

Proof. The first part follows from the fact that $\left|n - \varepsilon_J 2^J\right| < 2^J$. To prove the other two parts, we note that

$$|n| = \varepsilon_J n = 2^J - |\mathsf{value}(\varepsilon_{J-1}, \ldots, \varepsilon_0)|\,.$$

Since $(\varepsilon_{J-1}, \ldots, \varepsilon_0)$ is the alternating greedy expansion of the other type in each case (with most significant digit ε_k for some $k < J$), we get the estimates by induction. $\square$

At this point we note that Lemma 2 also inductively re-proves the uniqueness of both the balanced and the unbalanced alternating greedy expansions, since we have $J = \lfloor \log_2 |n| \rfloor + 1$ for the balanced and $J = \lceil \log_2 |n| \rceil$ for the unbalanced alternating greedy expansions and $\varepsilon_J = \operatorname{sign}(n)$ for both versions of the alternating greedy expansion.

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 $\boldsymbol{x} = (x^{(1)}, \ldots, x^{(d)})^T \in \mathbb{Z}^d$ be a column vector of integers. A *joint (signed binary) expansion of $\boldsymbol{x}$* is an array

$$\boldsymbol{\varepsilon} = (\varepsilon_j^{(k)})_{\substack{1 \leq k \leq d \\ j \in \mathbb{N}_0}} = (\boldsymbol{\varepsilon}_j)_{j \in \mathbb{N}_0} \in \{-1, 0, 1\}^{d \times \mathbb{N}_0}$$

such that only a finite number of $\boldsymbol{\varepsilon}_j$ is nonzero and such that

$$\boldsymbol{x} = \mathsf{value}(\boldsymbol{\varepsilon}) := \sum_{j \geq 0} \boldsymbol{\varepsilon}_j 2^j,$$

i.e., the rows $(\varepsilon_j^{(k)})_{j \geq 0}$ are expansions of $x^{(k)}$ 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.

If each row of $\boldsymbol{\varepsilon}$ is a unisigned binary or an alternating greedy expansion, we call $\boldsymbol{\varepsilon}$ a joint unisigned or alternating greedy expansion, respectively.

We recall from Solinas [15] that a joint expansion $\boldsymbol{\varepsilon}$ of $\boldsymbol{x} = (x^{(1)}, \ldots, x^{(d)})^T$ 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 $\boldsymbol{\varepsilon}$.

Thus we call a joint expansion $\boldsymbol{\varepsilon}$ of $\boldsymbol{x}$ a *minimal joint expansion of* $\boldsymbol{x}$, if it minimizes the joint Hamming weight over all joint expansions of $\boldsymbol{x}$.

In [4], we studied a class of minimal joint expansions, called the simple joint sparse form. As we will need its properties in the following sections, we summarize the relevant results of [4] in the following proposition.

Proposition 3. *Let $d \geq 1$ and $\boldsymbol{x} \in \mathbb{Z}^d$. Then there is a unique joint expansion $\boldsymbol{\varepsilon}$ of $\boldsymbol{x}$ satisfying the syntactical rule*

$$A_{j+1}(\boldsymbol{\varepsilon}) \supsetneq A_j(\boldsymbol{\varepsilon}) \text{ or } A_{j+1}(\boldsymbol{\varepsilon}) = \emptyset, \qquad j \geq 0, \tag{6}$$

where $A_j(\boldsymbol{\varepsilon}) = \{1 \leq k \leq d : \varepsilon_j^{(k)} \neq 0\}$. It is called the simple joint sparse form *of $\boldsymbol{x}$.*

The simple joint sparse form is a minimal joint expansion of $\boldsymbol{x}$. It can be computed from right to left by Algorithm 3. It has the property that

$$\text{among } d+1 \text{ consecutive columns of } \boldsymbol{\varepsilon}, \text{ there is at least one } \mathbf{0}. \tag{7}$$

Proof. Existence, uniqueness, and minimality have been proved in [4, Theorem 7]. Algorithm 3 is a reformulation of [4, Algorithm 2] in terms of the digits of an arbitrary joint expansion of $\boldsymbol{x}$. The property (7) is an easy consequence of (6). □

A comment on the epitheton "simple": Solinas [15] had proposed the so-called *joint sparse form* of two integers which also minimizes the joint Hamming weight before we started our paper [4]. Since in the case $d = 2$, Solinas' joint expansion needs a look-ahead of 2 and our expansion needs a look-ahead of 1 when computed by a transducer automaton from right to left, we called our expansion the "simple joint sparse form". Since both expansions are minimal joint expansions, their Hamming weight is equal. In the meantime, Solinas' joint sparse form has been generalized to higher dimensions by Proos [13].

For a joint expansion $\boldsymbol{\eta}$, we will denote the output of Algorithm 3 when reading $\boldsymbol{\eta}$ by $\mathsf{algSJSF}(\boldsymbol{\eta})$.

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 minimality of the output of our algorithms in Sections 4 and 5. The construction of our left-to-right algorithm in Section 4 will involve the use of Algorithm 3 for small blocks of joint alternating greedy expansions. Therefore, we will analyze the behaviour of Algorithm 3 in this special case.

Lemma 4. *Let $\boldsymbol{\eta}$ be a joint alternating greedy expansion of $\boldsymbol{x} \in \mathbb{Z}^d$ and let $\boldsymbol{\varepsilon} = \mathsf{algSJSF}(\boldsymbol{\eta})$. Then the following assertions hold.*

(1) *At all times, we have $|\varepsilon_j^{(k)}| \leq 1$ for all j and k in Algorithm 3, which implies that the first inner loop is always empty.*

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 $\boldsymbol{x}\in\mathbb{Z}^d$
Output: $(\varepsilon_j^{(k)})_{\substack{1\le k\le d\\0\le j\le J}}$ simple joint sparse form of $\boldsymbol{x}$
  $\boldsymbol{\eta}_{J+1} \leftarrow \mathbf{0}$
  $\boldsymbol{\varepsilon}_0 \leftarrow \boldsymbol{\eta}_0$
  $A_0 \leftarrow \{1\le k\le d : \varepsilon_0^{(k)} \text{ is odd}\}$
  for $j = 0$ to $J$ do
    {We have $A_j = \{1\le k\le d : \varepsilon_j^{(k)} \text{ is odd}\}$}
    $\boldsymbol{\varepsilon}_{j+1} \leftarrow \boldsymbol{\eta}_{j+1}$
    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
      {All components of $\boldsymbol{\varepsilon}_{j+1}$ can be made even}
      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
      {There are components of $\boldsymbol{\varepsilon}_{j+1}$ which cannot be made even, thus we generate as
      many odd components as possible}
      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
```

(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* $J := \max\{j \geq 0 : \boldsymbol{\eta}_j \neq \mathbf{0}\}$ *and assume that there is a* $1 \leq k \leq d$ *such that* $\eta_i^{(k)} = 0$ *for all* $i < J$*. Let* $(\boldsymbol{\varepsilon}'_{J-1}, \ldots, \boldsymbol{\varepsilon}'_0) = \mathsf{algSJSF}((\boldsymbol{\eta}_{J-1}, \ldots, \boldsymbol{\eta}_0))$*, then* $(\boldsymbol{\eta}_J, \boldsymbol{\varepsilon}'_{J-1}, \ldots, \boldsymbol{\varepsilon}'_0)$ *is a minimal 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 $i \leq j$ maximal such that $\eta_i^{(k)} \neq 0$ and $\eta_J^{(k)} = 0$ for $i < J \leq j$. By Equation (1), we have $\eta_i^{(k)} = -\eta_{j+1}^{(k)}$. For $i \leq J \leq j$ we get $\varepsilon_J^{(k)} \in \{-\eta_{j+1}^{(k)}, 0\}$. 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 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 4 (3), the length of $\boldsymbol{\varepsilon}'$ is appropriate. We always have $k \notin A_i$ for $i < J$ when computing $\mathsf{algSJSF}((\boldsymbol{\eta}_J, \boldsymbol{\varepsilon}'_{J-1}, \ldots, \boldsymbol{\varepsilon}'_0))$. Therefore, $k \in A_J \setminus A_{J-1}$ which implies that $\mathsf{algSJSF}$ would change $\eta_J^{(m)}$ for $m \in A_{J-1} \setminus A_J$, but at this step, no zero column can be produced, so $(\boldsymbol{\eta}_J, \boldsymbol{\varepsilon}'_{J-1}, \ldots, \boldsymbol{\varepsilon}'_0)$ is minimal, too. □

Now we can state the crucial proposition.

Proposition 5. *Let* $d \geq 1$*,* $\boldsymbol{x} \in \mathbb{Z}^d$ *and* $(\boldsymbol{\eta}_j)_{0 \leq j \leq J}$ *be the joint alternating greedy expansion of* $\boldsymbol{x}$*. Assume that the set*

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

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.,*

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

Then $\boldsymbol{\varepsilon}$ *is a minimal joint expansion of* $\boldsymbol{x}$*.*

Proof. First we note that by Lemma 4 (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 $\boldsymbol{x}$.

If $\varepsilon'^{(k)}_{I-1}$ is nonzero for some $1 \leq k \leq d$, say $\varepsilon'^{(k)}_{I-1} = 1$, then the integer $\sum_{j=0}^{I-1} \varepsilon'^{(k)}_j 2^j = \sum_{j=0}^{I-1} \eta_j^{(k)} 2^j$ 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 $(\eta_J^{(k)} \cdots \eta_I^{(k)} \varepsilon'^{(k)}_{I-1})$ satisfies (1).

Writing $\mathsf{algSJSF}((\boldsymbol{\eta}_J \cdots \boldsymbol{\eta}_I \boldsymbol{\varepsilon}'_{I-1})) = (\boldsymbol{\varepsilon}'''_j)_{I-1 \le j \le J}$, 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 4 (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 $i \ge I$. By Lemma 4 (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 online algorithm which can be realized by a transducer transforming the joint alternating greedy expansion of $\boldsymbol{x} \in \mathbb{Z}^d$ to an expansion of the same (and therefore minimal) joint Hamming weight as the simple joint sparse form from left to right. We note that it is impossible to compute the simple joint sparse form from left to right (the fractal structures discussed in [4] prove this; cf. also the example for $d = 1$ in [6]), we can only compute a joint expansion of the same Hamming weight. The algorithm presented in this section uses the right-to-left algorithm for subblocks of the joint alternating greedy expansion as its main ingredient. This makes the analysis simpler, but may sometimes lead to unnecessary operations in the computation. Therefore, we will refine this algorithm in Section 5.

Proposition 5 allows us to split the computation of a minimal joint expansion into several pieces. From (7) we also see that $J \ge I \ge J - d$ (if $J \ge d$). Therefore, we can use Algorithm 4 to compute a minimal joint expansion from left to right.

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

```
Input: $\boldsymbol{x} = (x^{(1)}, \dots, x^{(d)})^T \in \mathbb{Z}^d$
Output: joint expansion $\boldsymbol{\varepsilon}$ of $\boldsymbol{x}$ of minimal joint Hamming weight.
  $(\eta_j^{(k)})_{0 \le j \le J} \leftarrow$ joint alternating greedy expansion of $\boldsymbol{x}$
  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 unisigned binary expansions to a minimal joint expansion.

For $d = 2$, such a transducer automaton has been explicitly constructed in [5] (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 6. *Let $d \geq 1$. There is a transducer automaton to transform the unisigned joint expansion of an integer vector $\boldsymbol{x} \in \mathbb{Z}^d$ to a joint expansion of $\boldsymbol{x}$ 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 unisigned binary expansions. This is a considerable refinement of algMinJointViaSJSF, since it works based on simple column scanning and bitwise replacement without using algSJSF as an intermediate step. This leads to fewer replacements in the expansions. An example where Algorithm 4 and Algorithm 5 produce different outputs for the same input is given in Remark 10.

Algorithm 5 scans the unisigned binary expansions of the d integers from the most significant bit column $(J - 1)$ towards the least significant bit column (0), $d + 1$ columns at a time.

Algorithm 5 consists of two steps:

Step 1: Converting the unisigned binary input to the joint alternating greedy expansion.
Step 2: Making replacements on the joint alternating greedy expansion.

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

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

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

satisfying $\varepsilon_i^{(k)} \neq 0$.
C3: $\{i : j > i \geq \mathit{MinNextNonzeroLocation}\} = \{\mathit{RightmostNonzeroLocation}[k] : 1 \leq k \leq d \text{ and } k \notin \mathit{BitsAllZero}\}$

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 $x0\ldots0\overline{x}$ 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.

As an example, consider the input (already converted to its alternating greedy expansion and only the rightmost four columns are shown)

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

We start from $j = 3$. We note that $\varepsilon_3^{(0)} = \varepsilon_3^{(1)} = \varepsilon_3^{(2)} = 0$, thus *LeftmostIsNonzero* $= \emptyset$. **C1** is not satisfied so we move to $j = 2$. We have $\varepsilon_2^{(0)} = \varepsilon_2^{(1)} = 0$ and $\varepsilon_2^{(2)} = 1$ so *LeftmostIsNonzero* $= \{2\}$. **C1** is satisfied. Since *EndComputingAlternatingGreedy* $=$ $\max(2{-}3, 0) = 0$ and $\varepsilon_0^{(2)} = \bar{1}$, **C2** is satisfied. At this point we have *NextNonzeroLocation*$[2] =$ 0 and *MinNextNonzeroLocation* $= 0$. For row 0, *RightmostNonzeroLocation*$[0] = 1$ because $\varepsilon_1^{(0)} = \bar{1}$ and $\varepsilon_0^{(0)} = 0$. For row 1, $\varepsilon_2^{(1)} = \varepsilon_1^{(1)} = \varepsilon_0^{(1)} = 0$ so $1 \in$ *BitsAllZero*. For row 2, *RightmostNonzeroLocation*$[2] = 0$ because $\varepsilon_0^{(2)} = \bar{1}$. Therefore $\{$*RightmostNonzeroLocation*$[k]$: $1 \leq k \leq d$ and $k \notin$ *BitsAllZero*$\}$=$\{0,1\}$=$\{i : j > i \geq$ *MinNextNonzeroLocation*$\}$. Hence all conditions **C1**, **C2**, and **C3** are satisfied. We make replacements on columns 2, 1, and 0. The final result is

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

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

Lemma 7. *Let $\boldsymbol{x} \in \mathbb{Z}^d$ be a vector of integers with joint alternating greedy expansion $(\varepsilon_i^{(k)})_{0 \leq i \leq J}$ and let us assume that the leftmost column of the sliding window takes value K at some stage of the execution of Algorithm 5 (running on the unisigned expansion of $\boldsymbol{x}$). We set*

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

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

Proof. We prove the lemma by induction on K. We note that $I \geq K - d$ by (7).

If **C1** is violated, then we have $\mathsf{algSJSF}(\boldsymbol{\varepsilon}_K) = \mathbf{0}$, hence $I = K$ and $\boldsymbol{\varepsilon}'_K = \mathbf{0}$, which have both joint Hamming weight zero. So we can assume that **C1** holds. If **C2** is not satisfied, which means that there is a line k which has a nonzero bit in column K and zero bits in columns $K - 1$, …, $\max(K - d, 0)$, Lemma 4 (4) shows that column $\boldsymbol{\varepsilon}_K$ can be left unchanged. Decreasing K to $K - 1$ does not alter I, so $(\boldsymbol{\varepsilon}'_{K-1}, \dots, \boldsymbol{\varepsilon}'_I)$ has the same joint Hamming weight as $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_{K-1}, \dots, \boldsymbol{\varepsilon}_I))$ by induction. Thus $(\boldsymbol{\varepsilon}'_K, \dots, \boldsymbol{\varepsilon}'_I)$ has the same joint Hamming weight as $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_K, \dots, \boldsymbol{\varepsilon}_I))$ by Lemma 4 (4).

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

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

In this case it is easily seen that $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_i, \dots, \boldsymbol{\varepsilon}_{\textit{MinNextNonzeroLocation}})) = (\mathbf{0}, \mathbf{?}, \dots, \mathbf{?})$, where $\mathbf{?}$ stands for some uninteresting digit vector. We get $I \geq$ *MinNextNonzeroLocation*.

Algorithm 5 algMinJoint: Computing a Minimal Joint Expansion from the Unisigned Binary Expansion from Left to Right

```
Input: $(\eta_j^{(k)})_{\substack{1\le k\le d\\ 0\le j\le J-1}}$ joint unisigned binary expansion of $\boldsymbol{x} = (x^{(1)}, \dots, x^{(d)})^T \in \mathbb{Z}^d$
Output: $(\varepsilon_j^{(k)})_{\substack{1\le k\le d\\ 0\le j\le J}}$ signed binary expansion of $\boldsymbol{x}$ with minimum joint Hamming weight
  $\eta_J \leftarrow \mathbf{0}$
  $\eta_{-1} \leftarrow \mathbf{0}$
  $\mathit{StartComputingAlternatingGreedy} \leftarrow J$
  $j \leftarrow J$
  while $j \ge 0$ and $\mathit{StartComputingAlternatingGreedy} \ge 0$ do
    $\mathit{EndComputingAlternatingGreedy} \leftarrow \max(j-d, 0)$
    for $1 \le k \le d$ do
      for $\mathit{StartComputingAlternatingGreedy} \ge i \ge \mathit{EndComputingAlternatingGreedy}$ do
        $\varepsilon_i^{(k)} \leftarrow \eta_{i-1}^{(k)} - \eta_i^{(k)}$
      end for
    end for
    $\mathit{StartComputingAlternatingGreedy} \leftarrow \mathit{EndComputingAlternatingGreedy} - 1$
    $\mathit{LeftmostIsNonzero} \leftarrow \{1 \le k \le d : \varepsilon_j^{(k)} \ne 0\}$
    if C1 and C2 are satisfied then
      $\mathit{NextNonzeroLocation}[k] \leftarrow \max\{i : j > i$ and $\varepsilon_i^{(k)} \ne 0\}$ for each
      $k \in \mathit{LeftmostIsNonzero}$
      $\mathit{MinNextNonzeroLocation} \leftarrow \min\{\mathit{NextNonzeroLocation}[k] : k \in$
      $\mathit{LeftmostIsNonzero}\}$
      for $1 \le k \le d$ do
        if $\varepsilon_i^{(k)} \ne 0$ for some $j > i \ge \mathit{MinNextNonzeroLocation}$ then
          $\mathit{RightmostNonzeroLocation}[k] \leftarrow \min\{j > i \ge \mathit{MinNextNonzeroLocation} :$
          $\varepsilon_i^{(k)} \ne 0\}$
        end if
      end for
      $\mathit{BitsAllZero} \leftarrow \{1 \le k \le d : \varepsilon_i^{(k)} = 0$ for all $j > i \ge \mathit{MinNextNonzeroLocation}\}$
      if C3 is satisfied then
        for all $k \in \mathit{LeftmostIsNonzero}$ do
          $\varepsilon_i^{(k)} \leftarrow \varepsilon_j^{(k)}$ for each $i$ with $j-1 \ge i \ge \mathit{NextNonzeroLocation}[k]$
          $\varepsilon_j^{(k)} \leftarrow 0$
        end for
        $j \leftarrow \mathit{MinNextNonzeroLocation} - 1$
      else
        $j \leftarrow j-1$
      end if
    else
      $j \leftarrow j-1$
    end if
  end while
```

In the case $I > \mathit{MinNextNonzeroLocation}$, there is a $k \in \mathit{LeftmostIsNonzero}$ satisfying $\mathit{NextNonzeroLocation}[k] < I$. By Lemma 4 (4) this means that we can keep $\boldsymbol{\varepsilon}_K$ and decrease K to $K-1$ without changing I. Therefore, we are left with the case $I = \mathit{MinNextNonzeroLocation}$. Lemma 4 (2) says that $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_K, \dots, \boldsymbol{\varepsilon}_i, \dots, \boldsymbol{\varepsilon}_I))$ restarts after dealing with column i. By maximality of I, the output of $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_K, \dots, \boldsymbol{\varepsilon}_i))$ does not contain a $\mathbf{0}$. Therefore, once again, we can decrease K to $K-1$ without changing I and use the induction hypothesis.

Finally, we consider the case that all three conditions **C1**, **C2**, and **C3** are satisfied. Then $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_K, \dots, \boldsymbol{\varepsilon}_{\mathit{MinNextNonzeroLocation}})) = (\mathbf{0}, \mathbf{?}, \dots, \mathbf{?})$, 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 $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_K, \dots, \boldsymbol{\varepsilon}_I))$ cannot contain a zero column, contradiction. Therefore we have $I = \mathit{MinNextNonzeroLocation}$, and it is clear that $(\boldsymbol{\varepsilon}'_K, \dots, \boldsymbol{\varepsilon}'_I)$ and $\mathsf{algSJSF}((\boldsymbol{\varepsilon}_K, \dots, \boldsymbol{\varepsilon}_I))$ both have exactly one zero column.

□

Theorem 8. *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 5 and Lemma 7. □

Remark 9. Since the output of Algorithm 5 has minimal joint Hamming weight, the results in [5, Table 3] apply for our algorithm, too: If $H_{J,d}$ denotes the joint Hamming weight of a minimal joint expansion of d random J-digit integers, the asymptotic expected value $\mathbb{E}(H_{J,d})$ and the asymptotic variance $\mathbb{V}(H_{J,d})$ can be found in Table 1. By Hwang's [7] quasi-power theorem, we also know a central limit theorem, cf. [5].

Remark 10. 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 $\mathbf{0}$-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 4 (4)).

d	$\frac{1}{J}\mathbb{E}(H_{J,d})$	$\frac{1}{J}\mathbb{V}(H_{J,d})$
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	$\frac{21292819}{29681427}$	$\frac{5775339222194349670\,40}{26148954556492040001483}$

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

Lemma 11. *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 $J, \dots, J-d$ will be a zero column in the output of Algorithm 5.*

Proof. Since at least one of $d+1$ consecutive output columns of algSJSF is zero by (7), we have $I \geq J-d$ for the I defined in Lemma 7. By the same Lemma 7, at least one column (in the same range) of the output of Algorithm 5 must also be a $\mathbf{0}$. $\square$

Theorem 12. *Among $2d+1$ consecutive columns of the output of Algorithm 5, there is at least one $\mathbf{0}$.*

Proof. By Lemma 11, 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. $\square$

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 [14]. 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.

The w-NAF and therefore its left-to-right analogue presented here can also be used for computing multiples on elliptic curves: it corresponds to the so-called sliding window method where blocks of w digits of the binary expansion lead to one addition of the precomputed point corresponding to the block. Since this operation is also carried out from left to right, it is advantageous to compute a minimal such expansion from left to right.

Theorem 13. *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 expansion 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 (4) and (5). Since $\eta_t \neq 0$, we conclude that all $\varepsilon_t \in \mathcal{D}_w$.

To prove minimality, 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 $h+1$.

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

```
Input: Alternating greedy expansion $(\eta_J, \dots, \eta_0)$ of $n$; $w \geq 1$
Output: $\mathcal{D}_w$-expansion $\boldsymbol{\varepsilon}$ of $n$ of minimal Hamming weight
  $\boldsymbol{\varepsilon} \leftarrow \mathbf{0}$
  $j \leftarrow J$
  while $j \geq 0$ do
    if $\eta_j = 0$ then
      $j \leftarrow j - 1$
    else
      $t \leftarrow \max(j - w, 0)$
      while $\eta_t = 0$ do
        $t \leftarrow t + 1$
      end while
      $\varepsilon_t \leftarrow \sum_{\ell=t}^{j} \eta_\ell 2^{\ell-t}$
      $j \leftarrow j - w - 1$
    end if
  end while
```

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 (4) and (5), we have

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

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 (4) and (5) we have

$$\bigl| a_h + \varepsilon_t 2^{t-r_h} \bigr| \geq 2^{J-t-1} \cdot 2^{t-r_h} - 2^{J-w-1-r_h} = 2^{J-r_h-1} - 2^{J-w-1-r_h} \geq 2^w - 1. \tag{8}$$

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 (8). We conclude that $h' > h$. □

References

[1] R. Avanzi, *A Note on the Signed Sliding Window Integer Recoding and a Left-to-Right Analogue*, Selected Areas in Cryptography: 11th International Workshop, SAC 2004, Waterloo, Canada, August 9-10, 2004, Revised Selected Papers, Lecture Notes in Comput. Sci., vol. 3357, Springer-Verlag, Berlin, 2004, pp. 130–143.

[2] A. D. Booth, *A signed binary multiplication technique*, Quart. J. Mech. Appl. Math. **4** (1951), 236–240.

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

[4] 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.

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

[6] C. Heuberger, *Minimal expansions in redundant number systems: Fibonacci bases and greedy algorithms*, Period. Math. Hungar. **49** (2004), 65–89.

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

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

[9] 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, 2004, pp. II 365–368.

[10] M. Lothaire, *Algebraic combinatorics on words*, Encyclopedia of Mathematics and its Applications, vol. 90, Cambridge University Press, Cambridge, 2002.

[11] J. A. Muir and D. R. Stinson, *New minimal weight representations for left-to-right window methods*, Topics in Cryptology — CT-RSA 2005 The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14–18, 2005, Proceedings (A. J. Menezes, ed.), Lecture Notes in Comput. Sci., vol. 3376, Springer, Berlin, 2005, pp. 366–384.

[12] 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`.

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

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

[15] 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`.

(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.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.edu