Efficient Encoding Algorithm for Third-Order Spectral-Null Codes*

Vitaly Skachek Tuvi Etzion Ron M. Roth†

Computer Science Department
Technion — Israel Institute of Technology
Haifa 32000, Israel

e-mail: {vitalys,etzion,ronny}@cs.technion.ac.il

Abstract

An efficient algorithm is presented for encoding unconstrained information sequences into a third-order spectral-null code of length n and redundancy $9 \log_2 n + O(\log \log n)$. The encoding can be implemented using $O(n)$ integer additions and $O(n \log n)$ counter increments.

1 Introduction

Let F be the bipolar alphabet $\{+1, -1\}$. A word $\underline{x} = (x_1, x_2, \ldots, x_n)$ in F^n is a *k-th order spectral-null word* (at zero frequency) if the respective real polynomial $x_1 z + x_2 z^2 + \ldots + x_n z^n$ is divisible by $(z-1)^k$. We denote by $\mathcal{S}(n, k)$ the set of all k-th order spectral-null words in F^n. Any subset $\mathcal{C}$ of $\mathcal{S}(n, k)$ is called a *k-th order spectral-null code* of length n. The concatenation of any l words in $\mathcal{C}$ yields a word in $\mathcal{S}(nl, k)$; so, spectral-null codes can be used as block codes with a *redundancy* of $n - \log_2 |\mathcal{C}|$ bits (per block of length n).

*This work was supported by grant No. 95-522 from the United-States–Israel Binational Science Foundation (BSF), Jerusalem, Israel.

†Currently on sabbatical leave at Hewlett-Packard Laboratories, 1501 Page Mill Road, Palo Alto, CA 94306, USA.

The set $\mathcal{S}(n,k)$ is equivalently characterized by

$$\mathcal{S}(n,k) \;=\; \left\{ \, \underline{x} \in F^n \;:\; \sum_{j=1}^{n} (j+c)^{\ell} x_j = 0 \;, \quad \ell = 0,1,\ldots,k{-}1 \, \right\} , \tag{1}$$

where c is any real constant (see [2, Ch. 9], [6]).

First-order spectral-null codes are also known by the names *balanced codes*, *zero-disparity codes*, or *DC-free codes*. There is a known efficient encoding algorithm for these codes due to Knuth [4] (see also Al-Bassam and Bose [1]), resulting in codes with redundancy $\log_2 n + O(\log\log n)$, where n is the code length. By 'efficient' we refer to the time (and space) complexity of the encoding, which amounts in Knuth's algorithm to $O(n)$ increments/decrements of a $\lceil \log_2 n \rceil$-bit counter (memory trade-offs allow to reduce the redundancy to $\log_2 n + O(1)$). The redundancy of $\mathcal{S}(n,1)$ is $\frac{1}{2}\log_2 n + O(1)$, and such redundancy can be attained by enumerative coding [2]; in terms of complexity, however, enumerative coding is less efficient than Knuth's algorithm.

For the case $k = 2$, efficient coding algorithms were presented in [6] and [7] that have redundancy of $3\log_2 n + O(\log\log n)$ bits and time complexity that amounts to $O(n)$ additions of $O(\log n)$-bit integers. Enumerative coding already turns out to be impractical for this case [6]. The redundancy of $\mathcal{S}(n,2)$ is known to be $2\log_2 n + O(1)$ [7].

For higher orders k of spectral null, Karabed and Siegel presented in [3] a coding method based upon finite-state diagrams (see also Monti and Pierobon [5]). However, since the rate of their construction is strictly less than 1, the resulting redundancy is linear in the code length n. It follows that for any fixed k and sufficiently large n, this redundancy is significantly larger than the upper bound $O(2^k \cdot \log n)$ on the redundancy of $\mathcal{S}(n,k)$ which is proved in [6] by nonconstructive arguments. A recursive construction is presented in [6] whose redundancy is $O(n^{1-\epsilon_k})$, where $0 < \epsilon_k < 1$ and $\lim_{k\to\infty} \epsilon_k = 0$. Yet, this redundancy is still considerably larger than the actual redundancy of $\mathcal{S}(n,k)$.

In this work, we present an efficient algorithm for encoding unconstrained sequences into a third-order spectral-null code whose redundancy is logarithmic in the code length. More specifically, for code length n, the redundancy is $9\log_2 n + O(\log\log n)$ bits and the encoding complexity is $O(n)$ additions of $O(\log n)$-bit integers and $O(n\log n)$ increments/decrements of $\lceil \log_2 n \rceil$-bit counters.

2 A third-order spectral-null encoder

It was shown in [6] that the length of a third-order spectral-null word is divisible by 4, so the generated words will be of length $n = 2h$ for some even integer h. For such an n, we let m be the integer $\lceil \log_2 n \rceil = 1 + \lceil \log_2 h \rceil$. Our encoding scheme will map input words $\underline{y}$ of length $\geq 2h - 6m + 2$ over F, into words $\underline{x} \in \mathcal{S}(2h, 3)$ and $\underline{x}' \in \mathcal{S}(3m + O(\log m), 3)$; the concatenation of $\underline{x}$ and $\underline{x}'$, in turn, will form the output third-order spectral-null word.

We will use the definition of $\mathcal{S}(2h, 3)$ which is obtained from (1) by substituting $k = 3$ and $c = -h-1$. It will also be convenient to index the entries of $\underline{x}$ hereafter by $(x_{-h}, x_{-h+1}, \ldots, x_{h-1})$. For a real word $\underline{x}$ we define the *moments* of $\underline{x}$ by

$$\sigma_\ell(\underline{x}) \stackrel{\text{def}}{=} \sum_{j=-h}^{h-1} j^\ell \cdot x_j \ , \quad \ell = 0, 1, 2, \cdots \ .$$

Clearly, a word $\underline{x} \in F^n$ is in $\mathcal{S}(2h, 3)$ if and only if $\sigma_0(\underline{x}) = \sigma_1(\underline{x}) = \sigma_2(\underline{x}) = 0$.

Our encoding algorithm starts with a word $\underline{x}$ over $F \cup \{0\}$ which contains the input word $\underline{y}$ as a subword, and the remaining entries of $\underline{x}$ are initially set to zero. Next, the algorithm reduces to zero the absolute values of $\sigma_0(\underline{x})$, $\sigma_2(\underline{x})$, and $\sigma_1(\underline{x})$ (in that order), by a sequence of bit negations, bit shifts, and bit swaps, and by assigning values of F to the zero entries. The encoding ends by encoding recursively certain counters that were computed in the course of the algorithm, resulting in a word, $\underline{x}'$, which is concatenated with $\underline{x}$ to produce the final output third-order spectral-null word.

The algorithm makes use of the following index sets, all being subsets of $S = \{-h, -h+1, \ldots, h-1\}$:

- $S_{\text{C2}} = \{d_i\}_{i=0}^{2m-8} \cup \{e_i\}_{i=0}^{2m-8}$, where

$$(d_i, e_i) \ = \ \begin{cases} (-10 \cdot 2^{i/2}, -6 \cdot 2^{i/2}) & \text{if } i \text{ is even} \\ (-9 \cdot 2^{(i+1)/2}, -7 \cdot 2^{(i+1)/2}) & \text{if } i \text{ is odd} \end{cases} \ , \quad 0 \leq i \leq 2m-10 \ , \quad (2)$$

  $(d_{2m-9}, e_{2m-9}) = (\tau_1, \tau_2)$, and $(d_{2m-8}, e_{2m-8}) = (-\tau_1, 7)$, where τ_1 is the smallest odd integer in S which is at least $\sqrt{(h^2/2) + 49}$, and τ_2 is the largest odd integer in S which is at most $h/2$. We remove $\{d_i, e_i\}$ from S_{C2} if $d_i < -h$.[1]

- $S_{\text{C3}} = \{0, -3,\ 3, -5,\ 5,\ 6, -7, -9,\ 9,\ 10, -11,\ 12, -13,\ 14\}$.

- $S_{\text{D}} = \{\pm 2^i\}_{i=0}^{m-2}$.

We will assume hereafter that h is large enough, in which case the sets S_{C2}, S_{C3}, and S_{D} are pairwise disjoint.[2] We let S_0 be the union $S_{\text{C2}} \cup S_{\text{C3}} \cup S_{\text{D}}$. Note that $|S_0| \leq 2(2m-7) + 14 + 2(m-1) = 6m-2$.

[1]This can happen only for $i = 2m-10, 2m-11$. Nevertheless, in those cases where only $\{d_{2m-10}, e_{2m-10}\}$ can be removed, then $\{d_{2m-9}, e_{2m-9}\}$ is redundant as well. In fact, it turns out that we will need all the $2(2m-7)$ elements of S_{C2} only when h is close in value to a power of 2.

[2]As we show in the example of Section 4 and as pointed out in the previous footnote, some elements in S_{C2} may sometimes be excluded. This allows to have h as small as 18.

For a word $\underline{x}$ of length n and a subset A of S, we will use the notation $\langle \underline{x} \rangle_A$ for the subword of $\underline{x}$ which is indexed by A.

The algorithm is summarized in Figure 1.

Step A: Initialization of $\underline{x}$

Let $\langle \underline{x} \rangle_{S \setminus S_0} \leftarrow \underline{y}$ and $\langle \underline{x} \rangle_{S_0} \leftarrow \underline{0}$.

Step B: Reduction of $|\sigma_0(\underline{x})|$

For increasing values of indexes $j = -h, -h+1, \ldots$, negate x_j (i.e., let $x_j \leftarrow -x_j$) until $\underline{x}$ becomes balanced. Let j_{B} be the number of negations performed until this condition is met.

Step C: Reduction of $|\sigma_2(\underline{x})|$

Step C1: Shift cyclically the entries of $\langle \underline{x} \rangle_{S \setminus S_0}$, until the resulting $\underline{x}$ is such that $|\sigma_2(\underline{x})| \le h^2$. Let j_{C} be the smallest number of shifts applied until this condition is met.

Step C2: For decreasing values of $i = 2m-8, 2m-9, \ldots, 0$, reduce the value of $|\sigma_2(\underline{x})|$ by assigning $x_{d_i} = -x_{e_i} = -1$ if $\sigma_2(\underline{x}) \ge 0$ and $x_{d_i} = -x_{e_i} = 1$ otherwise.

Step C3: Let $\langle \underline{x} \rangle_{S_{\mathrm{C3}}} \leftarrow$ the row in Table 1 that corresponds to $|\sigma_2(\underline{x})|$. Negate $\langle \underline{x} \rangle_{S_{\mathrm{C3}}}$ if $\sigma_2(\underline{x}) \ge 0$.

Step D: Reduction of $|\sigma_1(\underline{x})|$

Step D1: For increasing values of indexes $j = 1, 2, \ldots$, swap x_j with x_{-j} until $|\sigma_1(\underline{x})| \le 2(h-1)$, and let j_{D} denote the number of swaps made until this condition is met.

Step D2: For decreasing values of $i = m-2, m-3, \ldots, 0$, reduce the value of $|\sigma_1(\underline{x})|$ by assigning $x_{2^i} = -x_{-2^i} = -1$ if $\sigma_1(\underline{x}) \ge 0$ and $x_{2^i} = -x_{-2^i} = 1$ otherwise.

Step E: Recursive encoding

Apply Step A–D recursively to the binary representation of $(j_{\mathrm{B}}, j_{\mathrm{C}}, j_{\mathrm{D}})$. Concatenate the resulting word with $\underline{x}$ as the final output of the encoder.

Figure 1: Third-order spectral-null encoder.

3 Analysis of the algorithm

3.1 Validity

We verify step by step that the algorithm indeed terminates with a third-order spectral-null word.

Step A ends with a word $\underline{x}$ that contains an even number of entries in F. Step B is essentially Knuth's algorithm applied to those entries. As shown in [4], there always exists

$\lvert\sigma_2(\underline{x})\rvert \setminus$ index	0	−3	3	−5	5	6	−7	−9	9	10	−11	12	−13	14
1	+	+	−	−	+	−	+	−	−	+	−	+	−	+
3	+	+	−	+	+	−	−	−	−	+	−	−	+	+
5	−	+	−	−	−	+	+	+	+	+	−	−	+	−
7	−	−	−	+	+	−	+	+	+	−	+	+	−	−
9	+	−	−	−	+	+	+	−	−	+	+	−	−	+
11	+	+	−	+	−	−	−	+	−	+	−	+	+	−
13	+	+	−	−	+	−	+	−	+	−	−	−	+	+
15	−	−	−	+	+	−	+	+	+	+	−	−	+	−
17	+	−	−	−	−	+	+	+	−	+	+	+	−	−
19	−	−	−	+	+	+	−	+	−	+	+	+	−	−
21	+	+	−	−	−	−	+	+	+	−	−	+	+	−
23	+	+	−	−	−	+	+	−	−	+	−	+	−	+
25	+	+	−	−	+	+	−	−	−	+	−	−	+	+
27	+	+	−	−	−	−	−	+	+	+	+	+	−	−
29	−	−	−	+	−	+	+	+	+	−	+	+	−	−
31	−	−	−	+	+	+	−	+	+	−	+	−	+	−
33	+	−	−	+	+	−	+	−	−	+	−	+	−	+
35	+	+	−	−	−	+	+	−	+	−	−	−	+	+
37	−	+	−	+	+	+	−	−	+	−	−	−	+	+
39	−	+	−	+	+	−	+	−	−	−	+	+	+	−
41	+	+	−	−	−	+	−	−	+	+	+	−	−	+
43	+	+	−	−	+	−	+	−	−	−	+	+	−	+
45	+	+	−	+	+	−	−	−	−	−	+	−	+	+
47	−	+	−	−	−	+	+	+	+	−	+	−	+	−
49	+	−	−	+	+	+	−	−	−	−	+	+	−	+
51	+	+	−	−	+	−	+	−	−	+	−	−	+	+
53	+	−	−	+	−	−	+	+	+	−	−	+	+	−
55	+	−	−	−	+	+	+	−	−	+	−	+	−	+
57	+	−	−	+	+	+	−	−	−	+	−	−	+	+
59	+	−	−	+	−	−	−	+	+	+	+	+	−	−
61	+	−	−	−	+	−	+	+	+	−	+	−	−	+
63	−	+	−	+	+	−	+	−	+	−	−	−	+	+

Table 1: Generating odd integers up to 63 by balanced assignments.

a prefix of $\underline{x}$ which, when negated, makes $\underline{x}$ balanced. Hence, the negation counter j_{B} is well-defined.

We turn now to Step C and first verify that the shift counter j_{C} is well-defined.

Lemma 3.1 *There is always a cyclic shift of $\langle \underline{x} \rangle_{S \setminus S_0}$ in Step C1 for which $|\sigma_2(\underline{x})| \le h^2$.*

Proof. Let $\underline{x}^{(0)}$ denote the value of $\underline{x}$ at the beginning of Step C1 and let $\underline{x}^{(s)} = (x^{(s)}_{-h}, x^{(s)}_{-h+1}, \ldots, x^{(s)}_{h-1})$ be the word obtained from $\underline{x}^{(0)}$ by s right cyclic shifts of $\langle \underline{x}^{(0)} \rangle_{S \setminus S_0}$ (note that $\langle \underline{x}^{(s)} \rangle_{S_0}$ remains zero for all s).

First, we show that $|\sigma_2(\underline{x}^{(s+1)}) - \sigma_2(\underline{x}^{(s)})| \le 2h^2$ for every $s \ge 0$. We say that location j in $\underline{x}^{(s)}$ contains a *sign change* if $x^{(s)}_j \ne x^{(s+1)}_j$. Let $j_1 < j_2 < \cdots < j_t$ be the locations of the sign changes in $\underline{x}^{(s)}$. It is easy to verify that

$$\left| \sigma_2(\underline{x}^{(s+1)}) - \sigma_2(\underline{x}^{(s)}) \right| \;=\; \left| 2 \sum_{i=1}^{t} (-1)^i \cdot j_i^2 \right| . \tag{3}$$

Let r be the smallest index i such that $j_i \geq 0$. Define $B^- = \sum_{i=1}^{r-1}(-1)^i \cdot j_i^2$ and $B^+ = \sum_{i=r}^{t}(-1)^i \cdot j_i^2$. Now, B^- is a sum of integers with alternating signs and decreasing absolute values, where the first integer in the sum (if any) is negative. Hence,

$$-h^2 \;\leq\; -j_1^2 \;\leq\; B^- \;\leq\; 0\,. \tag{4}$$

On the other hand, B^+ is a sum of integers with alternating signs and increasing absolute values. Furthermore, since t is even, the last integer in the sum is positive. Hence,

$$0 \;\leq\; B^+ \;\leq\; j_t^2 \;\leq\; (h{-}1)^2\,. \tag{5}$$

Combining (3), (4), and (5), we obtain,

$$\left|\sigma_2(\underline{x}^{(s+1)}) - \sigma_2(\underline{x}^{(s)})\right| \;=\; 2\cdot|B^- + B^+| \;\leq\; 2h^2\,. \tag{6}$$

Next, we observe that $\sum_{s=0}^{|S\setminus S_0|-1}\sigma_2(\underline{x}^{(s)}) = 0$. Indeed, since $\underline{x}^{(0)}$ is balanced, it follows that $\sum_{s=0}^{|S\setminus S_0|-1} x_j^{(s)} = 0$ for every $j \in S$. Hence,

$$\sum_{s=0}^{|S\setminus S_0|-1}\sigma_2(\underline{x}^{(s)}) \;=\; \sum_{s=0}^{|S\setminus S_0|-1}\sum_{j\in S} j^2\cdot x_j^{(s)} \;=\; \sum_{j\in S} j^2\cdot\sum_{s=0}^{|S\setminus S_0|-1} x_j^{(s)} \;=\; 0\,.$$

Therefore, there is a 'zero-crossing' value of s for which $\sigma_2(\underline{x}^{(s)})\cdot\sigma_2(\underline{x}^{(s+1)}) \leq 0$. By (6), for such an s we must have either $|\sigma_2(\underline{x}^{(s)})| \leq h^2$ or $|\sigma_2(\underline{x}^{(s+1)})| \leq h^2$. $\square$

Step C2 further reduces the absolute value of $\sigma_2(\underline{x})$ as follows.

Lemma 3.2 *The value of $\sigma_2(\underline{x})$ after Step C2 is an odd integer between -63 and 63.*

Proof. First note that

$$2(d_{i-1}^2 - e_{i-1}^2) \;\geq\; d_i^2 - e_i^2\,, \qquad i = 2m{-}8, 2m{-}9, \ldots, 1\,, \tag{7}$$

and $2(d_{2m-8}^2 - e_{2m-8}^2) \geq h^2$. Specifically, for the values in (2) we have $d_i^2 - e_i^2 = 2^{i+6}$ for $i \leq 2m{-}10$, and a simple check reveals that (7) holds also for $i \in 2m{-}8, 2m{-}9$ (if $d_{i-1} < -h$, then $\{d_{i-1}, e_{i-1}\}$ is removed from S_{C2}; nevertheless, it can be verified that (7) still holds if we replace (d_{i-1}, e_{i-1}) by the pair (d_r, e_r) of elements in S_{C2} with the largest index $r < i$). It follows that after iteration i in Step C2, the resulting absolute value of $\sigma_2(\underline{x})$ is bounded from above by $d_i^2 - e_i^2$. In particular, for $i = 0$, the value of $\sigma_2(\underline{x})$ is an integer between -64 and 64. Furthermore, at this stage, the only zero entries of $\underline{x}$ are those that are indexed by $S_{\mathrm{C3}} \cup S_{\mathrm{D}}$. Since $S \setminus (S_{\mathrm{C3}} \cup S_{\mathrm{D}})$ contains an odd number of odd indexes, it follows that $\sigma_2(\underline{x})$ must be odd. $\square$

The final reduction of $|\sigma_2(\underline{x})|$ to zero is done is Step C3, using Table 1. It can be readily checked that for $r = 1, 3, \ldots, 63$, the values in row r in the table contribute r to $\sigma_2(\underline{x})$ (we negate those values in Step C3 if the contribution needs to be $-r$). Note that neither of the changes made in Step C affects the value of $\sigma_0(\underline{x})$, which still remains zero.

We now turn to Step D. This step is very similar to "Phase A" of the second-order spectral-null encoder in [6, Section IV]). We show next that the swap counter j_{D} is well-defined.

Lemma 3.3 *There is always a word $\underline{x}$ obtained by less than h swaps in Step D1 for which $|\sigma_1(\underline{x})| \le 2(h-1)$.*

Proof. Let $\underline{x}^{[0]}$ denote the value of $\underline{x}$ at the beginning of Step D1 and let $\underline{x}^{[j]}$ be the word after the jth swap. First, it is easy to check that $|\sigma_1(\underline{x}^{[j+1]}) - \sigma_1(\underline{x}^{[j]})| \le 4(h-1)$ for all $j \ge 0$. Suppose we continue the swaps until $j = h-1$, and let $\underline{x}^{[h]}$ be the word obtained from $\underline{x}^{[h-1]}$ by negating the first entry (indexed by $-h$). In that case we will have $|\sigma_1(\underline{x}^{[h]}) - \sigma_1(\underline{x}^{[h-1]})| = 2h$ and

$$\sigma_1(\underline{x}^{[h]}) \;=\; -\sigma_1(\underline{x}^{[0]}) \;.$$

Hence, there must be a 'zero-crossing' index $j < h$ for which $\sigma_1(\underline{x}^{[j]}) \cdot \sigma_1(\underline{x}^{[j+1]}) \le 0$. For such a j we must have either $|\sigma_1(\underline{x}^{[j]})| \le 2(h-1)$ or $|\sigma_1(\underline{x}^{[j+1]})| \le 2(h-1)$. Furthermore, if the zero-crossing index is $j = h-1$, we have $|\sigma_1(\underline{x}^{[h-1]})| \le h$ or $|\sigma_1(\underline{x}^{[h]})| = |\sigma_1(\underline{x}^{[0]})| \le h$. $\square$

Turning to Step D2, it can be easily verified that after iteration i in that step, the resulting value of $|\sigma_1(\underline{x})|$ is bounded from above by 2^{i+1}. In particular, for $i = 0$, the value of $\sigma_1(\underline{x})$ is an integer between -2 and 2. The following lemma implies that $\sigma_1(\underline{x})$ is actually zero at this point.

Lemma 3.4 *For n divisible by 4 and every $\underline{w} \in F^n$,*

$$\sigma_1(\underline{w}) \;\equiv\; \sigma_2(\underline{w}) \pmod 4 \;.$$

Proof. Let $n = 2h$ and write $\underline{w} = (w_{-h}, w_{-h+1}, \ldots, w_{h-1})$. Then,

$$\begin{aligned}
\sigma_2(\underline{w}) - \sigma_1(\underline{w}) \;&=\; \textstyle\sum_{j=-h}^{h-1} j(j-1) \cdot w_j \\
&=\; \textstyle\sum_{l=-h/2}^{(h/2)-1} \Big((2l)(2l-1) \cdot w_{2l} + (2l+1)(2l) \cdot w_{2l+1}\Big) \\
&=\; \textstyle\sum_{l=-h/2}^{(h/2)-1} (2l)\Big((2l-1) \cdot w_{2l} + (2l+1) \cdot w_{2l+1}\Big) \;.
\end{aligned}$$

The result follows by observing that $(2l)\Big((2l-1) \cdot w_{2l} + (2l+1) \cdot w_{2l+1}\Big)$ is divisible by 4 for every l. $\square$

Neither of the changes made in Step D affects the values of $\sigma_0(\underline{x})$ or $\sigma_2(\underline{x})$, which still remain zero. Hence, at the end of Step D we will have $\sigma_1(\underline{x}) \equiv 0 \pmod 4$. And since $-2 \le \sigma_1(\underline{x}) \le 2$, it follows that $\sigma_1(\underline{x})$ is zero.

Finally, Step E is rather straightforward and is based on the fact that the concatenation of two k-th order spectral-null words yields a k-th order spectral-null word.

Decoding of $\underline{y}$ is done by first reconstructing the values j_{B}, j_{C}, and j_{D} from $\underline{x}'$. Once we have those three counters, we can reconstruct the values of $\underline{x}$ at the beginning of Steps D, C, and B (in that order).

3.2 Redundancy

We turn now to computing the redundancy of the code which is defined by the words generated by the algorithm for any given length.

There is no direct redundancy penalty in Step B, but we need m bits to represent the negation counter j_{B} which will contribute to the redundancy in Step E.

Steps C and D require $|S_0| \leq 6m{-}2$ bits to reduce $|\sigma_2(\underline{x})|$ and $|\sigma_1(\underline{x})|$ to zero. We also need m bits to represent the shift counter j_{C} and $m{-}1$ bits to represent the swap counter j_{D}.

In Step E, the encoding procedure is applied recursively to the $3m{-}1$ bits that represent $(j_{\mathrm{B}}, j_{\mathrm{C}}, j_{\mathrm{D}})$, thus generating a word $\underline{x}' \in \mathcal{S}(m', 3)$ of length $m' = 3m + O(\log m)$. Since $m = \lceil \log_2 n \rceil$, it follows that the total redundancy of the encoding scheme is $9 \log_2 n + O(\log \log n)$ bits. This expression will be an upper bound on the redundancy also if we replace n by the overall length, $n + m'$, of the output word.

3.3 Time and space complexity

Step B can be implemented by first computing the initial value of $\sigma_0(\underline{x})$ and then updating this value for each negation. This requires $O(n)$ increments/decrements of a $\lceil \log_2 n \rceil$-bit counter.

As for Step C, we need to have the value of $\sigma_2(\underline{x})$ for each cyclic shift in Step C1. Assuming that the squares of the elements between 1 and h are pre-computed in a table, the initial value of $\sigma_2(\underline{x})$ in this step can be found in $O(n)$ additions of $O(\log n)$-bit integers. Now, let $\hat{\underline{x}}$ denote the word obtained from $\underline{x}$ by one right cyclic shift of $\langle \underline{x} \rangle_{S \setminus S_0}$, and let $\tilde{\underline{x}}$ be the word obtained from $\underline{x}$ by one right cyclic shift of the *whole* word $\underline{x}$. We describe next how $\sigma_\ell(\hat{\underline{x}})$ can be computed efficiently from $\sigma_\ell(\underline{x})$, $\ell = 1, 2$. Step C1 will then proceed iteratively by making $\hat{\underline{x}}$ the new value of $\underline{x}$.

Noting that $\sigma_0(\underline{x}) = 0$, it is easy to verify that

$$\sigma_1(\tilde{\underline{x}}) \;=\; \sigma_1(\underline{x}) - 2h \cdot x_{h-1} \quad \text{and} \quad \sigma_2(\tilde{\underline{x}}) \;=\; \sigma_2(\underline{x}) + 2\sigma_1(\underline{x}) \;.$$

Therefore, once we have $\sigma_1(\underline{x})$ and $\sigma_2(\underline{x})$, it is straightforward to compute $\sigma_1(\tilde{\underline{x}})$ and $\sigma_2(\tilde{\underline{x}})$.

Let S_1 denote the set of all indexes $j \in S_0$ such that $j-1 \in S \setminus S_0$ (when $j = -h$, the index $j-1$ should read $h-1$). For an index $j \in S_1$, let $\hat{\jmath}$ denote the smallest index in $S \setminus S_0$ which is larger than j (if no such index exists, then $\hat{\jmath}$ is defined as the smallest index in $S \setminus S_0$). For $\ell = 1, 2$, define

$$\alpha_\ell(\underline{x}) \;=\; \sum_{j \in S_1} (\hat{\jmath}^\ell - j^\ell) \cdot x_{j-1} \; .$$

It can be readily verified that

$$\sigma_\ell(\hat{\underline{x}}) \;=\; \sigma_\ell(\tilde{\underline{x}}) + \alpha_\ell(\underline{x}) \; , \qquad \ell = 1, 2 \; .$$

The expressions $\alpha_\ell(\underline{x})$ can be computed using $O(\log n)$ additions of $O(\log n)$-bit integers. The following discussion outlines how the computation of $\alpha_\ell(\underline{x})$ can be accelerated further through the use of small look-up tables.

Let $S_1 = \bigcup_t S_1(t)$ be a partition of S_1 into $O(1)$ subsets $S_1(t)$, each of size less than m. For each subset $S_1(t)$, construct a look-up table for computing the expression

$$\alpha_\ell\Big(\langle \underline{x} \rangle_{S_1(t)}\Big) \;=\; \sum_{j \in S_1(t)} (\hat{\jmath}^\ell - j^\ell) \cdot x_{j-1} \; ,$$

as a function of the entries $x_{j-1}, j \in S_1(t)$. Each look-up table consists of less than n entries and each entry contains an $O(\log n)$-bit integer. Note that these look-up tables can be computed in time $O(n)$ and that they depend on n, but not on the encoded word. In order to access the bits $x_{j-1}, j \in S_1(t)$ within $\langle \underline{x} \rangle_{S \setminus S_0}$, we will use $|S_1(t)|$ pointers (counters) that will be decremented after each shift. (In hardware implementations, we can instead store $\langle \underline{x} \rangle_{S \setminus S_0}$ in a shift-register.) Once we have computed the $O(1)$ expressions $\alpha_\ell(\langle \underline{x} \rangle_{S_1(t)})$, we obtain $\alpha_\ell(\underline{x})$ as their sum. Note that this computation of $\sigma_2(\underline{x})$ allows us to find j_{C} without actually shifting $\langle \underline{x} \rangle_{S \setminus S_0}$. This makes the computation efficient also in software implementations.

Steps C2, C3, and D are rather straightforward and can be implemented using $O(n)$ integer additions. Hence, the overall time and space complexity of our encoding algorithm is as follows:

- $O(n)$ additions of $O(\log n)$-bit integers,
- $O(n)$ accesses to $O(1)$ tables, each of size $< n$, and —
- $O(n)$ increments/decrements of $O(\log n)$ counters, each $\lceil \log_2 n \rceil$ bits long.

4 Example

We consider here the case $n = 60$ (for such a small value of n the redundancy is relatively big, so this example is given only for the purpose of illustrating the encoding steps). In this case $h = 30$ and $m = 6$, and the set S_{C2} is given by

$$S_{\mathrm{C2}} \;=\; \{-10, -18, -20, -23\} \cup \{-6, -14, -12,\ 7\}\ ,$$

where $\tau_1 = 23$. Note that we have excluded the elements $\{d_3, e_3\} = \{\tau_1, \tau_2\} = \{23, 15\}$ from S_{C2} since they will not be required in Step C2: The value of $d_2^2 - e_2^2 = (-20)^2 - (-12)^2 = 256$ is already greater than half the value of $d_4^2 - e_4^2 = (-23)^2 - 7^2 = 480$. The set S_{C3} is of size 14 and S_{D} is given by $\{\pm 2^i\}_{i=0}^{4}$. Hence, $|S_0| = 32$.

Suppose that the input word $\underline{y}$ of length $n - |S_0| = 28$ is given by

$$++++--+++++++-+-+++--++-+--+$$

After embedding $\underline{y}$ in $\underline{x}$ in Step A, we obtain the word

$$++++--+0++0+0+0+00000000000000\overset{\downarrow}{0}0000000000+0-0+0-+++--++-+--+$$

for which $\sigma_0(\underline{x}) = 10$ (the arrow points at the entry indexed by 0). The sequence of negations in Step B generates words $\underline{x}$ with $\sigma_0(\underline{x}) = 8, 6, 4, 2, 4, 6, 4, 4, 2, 0$. Hence, $j_{\mathrm{B}} = 10$, and Step B ends with

$$----++-0--0+0+0+00000000000000\overset{\downarrow}{0}0000000000+0-0+0-+++--++-+--+$$

For this word we have $\sigma_2(\underline{x}) = -2047$, and when applying the cyclic shifts in Step C1 we produce words $\underline{x}$ with $\sigma_2(\underline{x}) = -1853, -1755, -1357,$ and -625. The last value corresponds to

$$+--+---0-+0+0-0-00000000000000\overset{\downarrow}{0}0000000000-0+0+0++-+-+++--++-$$

which is the first word in this step with $|\sigma_2(\underline{x})| \le h^2 = 900$; so, $j_{\mathrm{C}} = 4$. The assignment of values to the entries indexed by S_{C2} in Step C2 results in

$$+--+---+-+++--0-+0-0+000-00000\overset{\downarrow}{0}000000-000-0+0+0++-+-+++--++-$$

with $\sigma_2(\underline{x}) = 47$. In Step C3, we fill in the entries indexed by S_{C3} with the negated entries of the row that corresponds to 47 in Table 1. This produces the word

$$+--+---+-+++--0-+---+-0--+0-00\overset{\downarrow}{+}00+0+--0-+-++++0++-+-+++--++-$$

Step D1 starts with $\sigma_1(\underline{x}) = 174$ and then continues with iterated swaps that generate words $\underline{x}$ with $\sigma_1(\underline{x}) = 194, 194, 182$ (9 iterations)$, 134, 82, 82,$ and 22. The last value corresponds to

$$+--+---+-+++--0++++-+-0--+0+00\overset{\downarrow}{+}00-0+--0-+---+-0++-+-+++--++-$$

and this word is the first to occur in this step with $|\sigma_1(\underline{x})| \le 2(h-1) = 58$, and so $j_{\mathrm{D}} = 15$. Step D2 fills in the entries indexed by S_{D} to produce the word

$$+--+---+-+++--+++++-+----+++-+\overset{\downarrow}{+}-+--+--+-+---+--++-+-+++--++-$$

for which we have $\sigma_0(\underline{x}) = \sigma_1(\underline{x}) = \sigma_2(\underline{x}) = 0$.

There are ways to make the counting of the negations/swaps more economical: We can disregard zero entries in $\underline{x}$ when counting the negations in Step B, and we can skip index pairs $(-j, j)$ with $x_{-j} = x_j$ when counting swaps in Step D (in which case the swaps become in effect negations of x_{-j} and x_j whenever $x_{-j} \neq x_j$).

Finally, the counters $(j_{\mathrm{B}}, j_{\mathrm{C}}, j_{\mathrm{D}})$ are coded into up to $3 \cdot 6 - 1 = 17$ bits and undergo a recursive encoding in Step E.

References

[1] S. Al-Bassam, B. Bose, *On balanced codes, IEEE Trans. Inform. Theory,* IT-36 (1990), 406–408.

[2] K.A.S. Immink, *Coding Techniques for Digital Recorders,* London: Prentice-Hall, 1991.

[3] R. Karabed, P.H. Siegel, *Matched spectral-null codes for partial-response channels, IEEE Trans. Inform. Theory,* IT-37 (1991), 818–855.

[4] D.E. Knuth, *Efficient balanced codes, IEEE Trans. Inform. Theory,* IT-32 (1986), 51–53.

[5] C.M. Monti, G.L. Pierobon, *Codes with a multiple spectral null at zero frequency, IEEE Trans. Inform. Theory,* IT-35 (1989), 463–472.

[6] R.M. Roth, P.H. Siegel, A. Vardy, *High-order spectral-null codes: Constructions and bounds, IEEE Trans. Inform. Theory,* IT-40 (1994), 1826–1840.

[7] L.G. Tallini, S. Al-Bassam, B. Bose, *On efficient high-order spectral-null codes, Proceedings of IEEE International Symposium On Information Theory,* Sept. 1995, p. 144, Whistler, BC, Canada.