

## REFERENCES

- [1] J. D. Marca and N. Jayant, "An algorithm for assigning binary indices to the code vectors of a multi-dimensional quantizer," in *Proc. IEEE Int. Conf. Communications*, Seattle, WA, June 1987, vol. 1, pp. 1128–1132.
- [2] N. Farvardin, "A study of vector quantization for noisy channels," *IEEE Trans. Inform. Theory*, vol. 36, pp. 799–809, July 1990.
- [3] H.-S. Wu and J. Barba, "Index allocation in vector quantisation for noisy channels," *Electron. Lett.*, vol. 29, pp. 1318–1320, July 1993.
- [4] K. Zeger and A. Gersho, "Pseudo-gray coding," *IEEE Trans. Commun.*, vol. 38, pp. 2147–2158, Dec. 1990.
- [5] P. Knagenhjelm and E. Agrell, "The Hadamard transform—A tool for index assignment," *IEEE Trans. Inform. Theory*, vol. 42, pp. 1139–1151, July 1996.
- [6] D. J. Goodman and T. J. Moulshley, "Using simulated annealing to design transmission codes for analogue sources," *Electron. Lett.*, vol. 24, pp. 617–618, May 1988.
- [7] Y. Yamaguchi and T. S. Huang, "Optimum binary fixed-length block codes," Quarterly Progress Rep. 78, MIT, Cambridge, MA, July 1965.
- [8] T. S. Huang, "Optimum binary code," Quarterly Progress Rep. 82, MIT, Cambridge, MA, July 1966.
- [9] T. S. Huang *et al.*, "Design considerations in PCM transmission of low-resolution monochrome still pictures," *Proc. IEEE*, vol. 55, pp. 331–335, Mar. 1967.
- [10] T. R. Crimmins, H. M. Horwitz, C. J. Palermo, and R. V. Palermo, "Minimization of mean-square error for data transmitted via group codes," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 72–78, Jan. 1969.
- [11] S. W. McLaughlin, D. L. Neuhoff, and J. J. Ashley, "Optimal binary index assignments for a class of equiprobable scalar and vector quantizers," *IEEE Trans. Inform. Theory*, vol. 41, pp. 2031–2037, Nov. 1995.
- [12] A. Méhes and K. Zeger, "Binary lattice vector quantization with linear block codes and affine index assignments," *IEEE Trans. Inform. Theory*, vol. 44, pp. 79–95, Jan. 1998.
- [13] R. E. Totty and G. C. Clark, "Reconstruction error in waveform transmission," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 336–338, Apr. 1967.
- [14] A. Méhes and K. Zeger, "Redundancy free codes for binary discrete memoryless channels," in *Proc. Conf. Information Science and Systems*, Princeton, NJ, 1994, pp. 1057–1062.
- [15] —, "On the performance of affine index assignments for redundancy free source-channel coding," in *Proc. Data Compression Conf.*, Snowbird, UT, Mar. 1995, p. 433.
- [16] —, "Affine index assignments for binary lattice quantization with channel noise," in *ISIT-95*, Whistler, B.C., Canada, Sept. 1995, p. 377.
- [17] R. Hagen and P. Hedelin, "Robust vector quantization by a linear mapping of a block code," *IEEE Trans. Inform. Theory*, vol. 45, pp. 200–218, Jan. 1999.
- [18] —, "Design methods for VQ by linear mappings of block codes," in *Proc. IEEE Int. Symp. Inform. Theory*, Trondheim, Norway, June 1994, p. 241.
- [19] —, "Robust vector quantization by linear mappings of block-codes," in *Proc. IEEE Int. Symp. Inform. Theory*, San Antonio, TX, Jan. 1993, p. 171.

## Efficient Code Constructions for Certain Two-Dimensional Constraints

Roman Talyansky, Tuvii Etzion, *Member, IEEE*,  
and Ron M. Roth, *Senior Member, IEEE*

**Abstract**—Efficient encoding algorithms are presented for two types of constraints on two-dimensional binary arrays. The first constraint considered is that of  $t$ -conservative arrays, where each row and each column has at least  $t$  transitions of the form '0'  $\rightarrow$  '1' or '1'  $\rightarrow$  '0.' The second constraint is that of two-dimensional DC-free arrays, where in each row and each column the number of '0's equals the number of '1's.

**Index Terms**—Balanced arrays, conservative arrays, two-dimensional constraints, two-dimensional DC-free arrays, two-dimensional runlength-limited constraints.

### I. INTRODUCTION

Recent developments in optical storage—especially in the area of holographic memory—are attempting to increase the recording density by exploiting the fact that the recording device is a *surface*. Under this new model, the recorded data is regarded as two-dimensional (2-D), as opposed to the track-oriented one-dimensional (1-D) recording paradigm [5], [22]. The new approach, however, introduces new types of constraints on the data—those now become 2-D rather than 1-D.

One-dimensional constraints were extensively studied, and there are several known methodologies for designing codes for such constraints; see, for instance, [15], [16], and [24]. On the other hand, our knowledge of 2-D constraints is much less profound. This might be attributed in part to the fact that the practical interest in those constraints has been risen only recently; however, it seems that the main reason for such a lack of knowledge is the provable difficulty of 2-D constraints compared to the 1-D case [4], [23]. Nevertheless, there have been several results reported on 2-D runlength-limited coding [7], [8], coding for holographic memory [2], [3], [27], and multitrack modulation coding [14], [17], [18]. Reference [9] deals with the computation of the capacity of 2-D constraints, and bounds on the capacity of certain specific constraints are presented in [6] and [28].

We next describe briefly two applications of 2-D constrained codes: holographic storage and barcodes. In holographic recording, data is stored optically in the form of 2-D pages. Each data page is a pattern of '0's and '1's, represented by dark and light spots, respectively. What is actually stored is the interference pattern between an optical representation of the data page and a so-called reference beam. Several holograms can be stored in the same physical volume, each being encoded with a distinct reference beam. To increase the reliability of the holographic recording system, the patterns of '0's and '1's need to satisfy certain modulation constraints. One example of such a constraint is avoiding long periodic stretches of dark or light spots in both dimensions [5], [27]. In addition, it is desirable to use coding techniques that do not permit a large imbalance

Manuscript received March 8, 1998. This work was supported in part by the United States—Israel Binational Science Foundation (BSF), Jerusalem, Israel, under Grant 95-522.

R. Talyansky is with the IBM Haifa Research Laboratory, Matam—Advanced Technology Center, Haifa 31905, Israel.

T. Etzion is with the Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel.

R. M. Roth is with the Computer Science Department, Technion—Israel Institute of Technology, Haifa 32000, Israel (e-mail: ronny@cs.technion.ac.il).

Communicated by A. Barg, Associate Editor for Coding Theory.

Publisher Item Identifier S 0018-9448(99)01420-0.

between ‘0’s and ‘1’s so that, during recording, the amount of signal light be independent of the data content [27]. More information on holographic recording, including specific coding methods for this medium, can be found in [2], [3], [5], [10], [11], [20], [25], and [27].

Two-dimensional constraints are also found in barcodes, which are widely used in retail stores and on production lines. In the current barcode standard, the constraint is still 1-D and is defined by means of an upper bound on the runlengths of ‘0’s and ‘1’s, with the additional requirement that the number of runs is fixed within a given block. In order to increase the number of possible barcodes, 2-D barcodes are proposed for future applications [19], where the physical read head can be a laser device or a charge-coupled-device, and similar runlength constraints occur both horizontally and vertically.

In the sequel, we use the following terms. A binary word is *t-good* if it contains at least  $t$  transitions of the form ‘0’  $\rightarrow$  ‘1’ or ‘1’  $\rightarrow$  ‘0’ (unlike [27], we do not extend the word cyclically when counting the transitions). A word that is not *t-good* will be called *t-bad*. A binary word is *blank* if it is 1-bad, namely, it is either all-zero or all-one. A binary word is *balanced* if it contains the same number of ‘0’s and ‘1’s.

In this work, we consider the coding problem of binary  $n_1 \times n_2$  arrays that satisfy the following types of 2-D constraints.

- **Conservative arrays:** Each row and each column in the array is nonblank. In other words, every row and every column has at least one transition ‘0’  $\rightarrow$  ‘1’ or ‘1’  $\rightarrow$  ‘0’ [27]. When the plane (e.g., the recording surface) is tiled by such arrays, the longest vertical (respectively, horizontal) run of identical bits will not exceed  $2n_1 - 2$  (respectively,  $2n_2 - 2$ ). More generally, we consider *t-conservative arrays* in which every row and every column is *t-good*. We denote the set of all *t-conservative arrays* of order  $n_1 \times n_2$  by  $\mathcal{C}(n_1 \times n_2, t)$ .
- **2-D DC-free arrays:** Each row and each column in the array is a balanced word (here  $n_1$  and  $n_2$  are assumed to be even). Two-dimensional DC-free arrays are an extreme case of conservative arrays in which every row and column is nonblank by virtue of the balancing property. We denote the set of all  $n_1 \times n_2$  2-D DC-free arrays by  $\mathcal{A}(n_1 \times n_2)$ .

Let  $\mathcal{X}$  be a set of binary  $n_1 \times n_2$  arrays. The *redundancy* of  $\mathcal{X}$ , denoted  $\rho(\mathcal{X})$ , is defined by

$$\rho(\mathcal{X}) = n_1 n_2 - \log_2 |\mathcal{X}|.$$

Let  $\mathcal{S}$  be a set of binary  $n_1 \times n_2$  arrays, e.g.,  $\mathcal{S} = \mathcal{C}(n_1 \times n_2, t)$  or  $\mathcal{S} = \mathcal{A}(n_1 \times n_2)$ . A (*lossless block*) *encoder* for  $\mathcal{S}$  is a one-to-one mapping  $\phi$  from the set  $\{0, 1\}^k$  into  $\mathcal{S}$ . The redundancy of  $\phi$  is  $n_1 n_2 - k$ , which, clearly, is also the redundancy of the set  $\mathcal{X} = \phi(\{0, 1\}^k)$  (i.e., the set of actual images under  $\phi$ ). The task of designing an encoder for a given constraint is finding a mapping  $\phi$  that has the smallest possible redundancy and can be efficiently implemented.

We will assume hereafter that  $n_1 \geq n_2$ . Define

$$\Delta(n_1 \times n_2, t) = n_2 - (t-1)\log_2 n_2 - \log_2 n_1.$$

In [27], an encoder was presented for *t-conservative arrays*, and the redundancy of that encoder is  $\lceil \log_2(n_1 + 1) + \log_2(n_2 + 1) \rceil$ . On the other hand, it is easy to verify the upper bound (see Appendix)

$$\rho(\mathcal{C}(n_1 \times n_2, t)) \leq -\log_2(1 - 4 \cdot 2^{-\Delta(n_1 \times n_2, t)}). \quad (1)$$

Therefore,  $\rho(\mathcal{C}(n_1 \times n_2, t)) \rightarrow 0$  when  $\Delta(n_1 \times n_2, t) \rightarrow \infty$ ; in particular, when

$$\Delta(n_1 \times n_2, t) = n_2 - (t-1)\log_2 n_2 - \log_2 n_1 \geq 3 \quad (2)$$

there is a subset  $\mathcal{X} \subseteq \mathcal{C}(n_1 \times n_2, t)$  with  $\rho(\mathcal{X}) \leq 1$ .

In Section II, we introduce such a set  $\mathcal{X}$  by presenting an encoder for  $\mathcal{C}(n_1 \times n_2, t)$  with redundancy of one bit. Furthermore, our encoder lends itself to efficient implementations.

Two-dimensional DC-free arrays are treated in Section III. We present an efficient encoder for  $\mathcal{A}(n_1 \times n_2)$  with redundancy

$$n_2 \log_2 n_1 + \frac{1}{2} n_1 \log_2 n_2 + O(n_1 + n_2 \log \log n_1) \quad (3)$$

(assuming that  $n_1 \geq n_2$ ). Now, it is known that the number of balanced words of length  $n_2$  is bounded from above by  $2^{n_2} / \sqrt{(\pi/2)n_2}$  [12]. Hence, it follows that  $\rho(\mathcal{A}(n_1 \times n_2))$  lies between  $\frac{1}{2}n_1(\log_2 n_2 + \log_2(\pi/2))$  and (3). We mention that it has been recently shown in [21] that  $\rho(\mathcal{A}(n_1 \times n_2))$  equals  $\frac{1}{2}(n_1 \log_2 n_2 + n_2 \log_2 n_1) + O(n_1)$ .

## II. ENCODING *t*-CONSERVATIVE ARRAYS

We present here an efficient encoder for *t-conservative arrays* that has redundancy 1 when  $n_1$ ,  $n_2$ , and  $t$  satisfy

$$n_1 \geq n_2 \geq 3 + \lceil \log_2(n_1 + n_2) \rceil + (2t-1)\lceil \log_2(n_2 + 1) \rceil. \quad (4)$$

The encoder accepts as input  $n_1 n_2 - 1$  unconstrained bits which are assumed to be arranged in an  $n_1 \times n_2$  array  $\Gamma$  where the upper-leftmost entry is reserved for the redundancy bit. That bit is initially reset to ‘0.’

The main idea of our encoder is “recycling *t*-bad rows,” i.e., we take advantage of the fact that *t*-bad rows, which need to be eliminated from  $\Gamma$ , do not contain much information in the first place; since the content of a row is determined uniquely by its first bit and its transition locations, each *t*-bad row carries no more than  $1 + (t-1)\log_2 n_2$  bits of information, and this number will typically be much smaller than  $n_2$  if condition (4) is satisfied (e.g., when  $t = 1$ , the bad rows are blank and carry only one bit of information). Therefore, we reuse those *t*-bad rows to record changes made on  $\Gamma$  that eliminate the *t*-bad rows and columns.

We introduce the following relation between binary  $n_1 \times n_2$  arrays. Let  $\Gamma_1$  and  $\Gamma_2$  be such arrays and, for  $i = 1, 2$ , let  $\tau_i$  be the smallest integer for which  $\Gamma_i$  is not  $\tau_i$ -conservative. We say that  $\Gamma_1$  is *more conservative* than  $\Gamma_2$  if either: 1)  $\tau_1 > \tau_2$  or 2)  $\tau_1 = \tau_2$  and there are less  $\tau_1$ -bad rows and columns in  $\Gamma_1$  than in  $\Gamma_2$ .

Our encoder is described through the algorithm CONSERVATIVE in Fig. 1. When the contents of  $\Gamma$  (with its upper-leftmost bit equaling ‘0’) is already a *t-conservative array*, no changes are made and the output of CONSERVATIVE is  $\Gamma$  itself (see Step 2). Otherwise, the upper-leftmost bit is set to ‘1’ (Step 3), as an indication to the decoding side that  $\Gamma$  undergoes changes that will make it *t-conservative*. Those changes will be recorded in the *t*-bad rows, which will be made into a linked list, with one formerly *t*-bad row pointing to the next. We start this linked list in the field  $\mathbf{u}$ , which consists of bits 2 through  $n_2$  in the first row of  $\Gamma$ . Since initially the field  $\mathbf{u}$  is not necessarily *t*-bad, its contents are swapped with one of the *t*-bad rows or columns in  $\Gamma$  (Step 4). The linked list is constructed in Step 5, where we code the ‘nil’ pointer by ‘0’ (say) and any other row pointer by a word that starts with ‘1,’ followed by  $\lceil \log_2(n_1 + 1) \rceil$  bits. In addition to that pointer, each formerly *t*-bad row contains the locations of its original transitions. By judiciously selecting the words used to code the  $t-1$  transition locations (each such word being  $\lceil \log_2(n_2 + 1) \rceil$  bits long), we can guarantee that Step 5 produces at least  $t$  transitions in every *t*-bad row. Step 6 concludes the treatment of the *t*-bad rows by recording the pointer of the row or column that was swapped in Step 4.

As an example, suppose that  $t = 3$  and that initially rows 4 and 9 are 3-bad; e.g., the contents of  $\Gamma$  at rows 1, 4, and 9 are given by

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & \cdots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & \cdots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & \cdots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \begin{matrix} \leftarrow 1 \\ \\ \leftarrow 4 \\ \\ \leftarrow 9 \end{matrix}$$

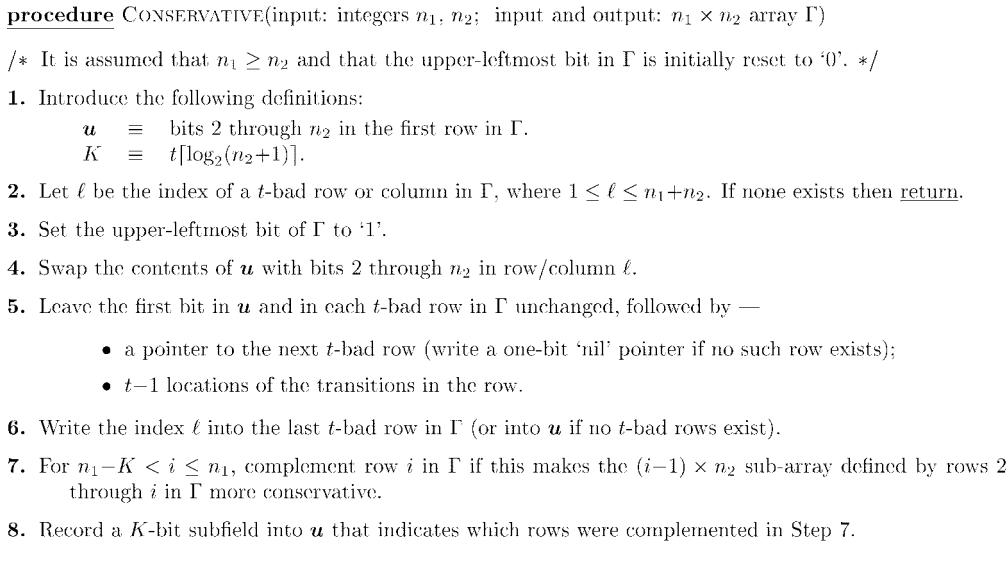
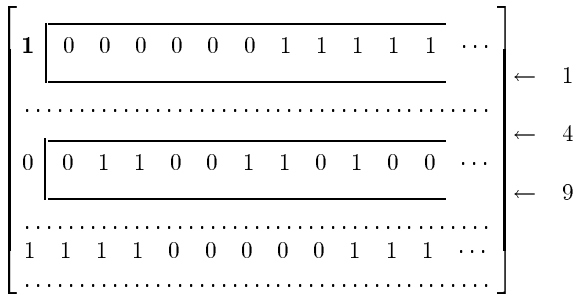
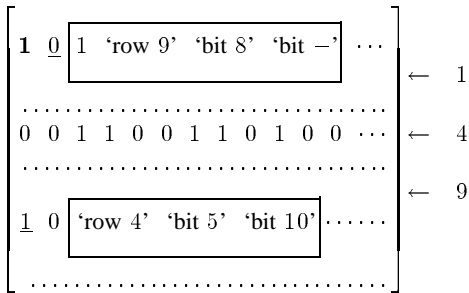


Fig. 1. Encoding algorithm for conservative arrays.

Following Step 4 we obtain



with  $\ell = 4$ . In Steps 5 and 6 we insert pointers as follows:



(The marked fields—which are not shown to scale—contain the row pointers and transition locations, and the underlined bits are those that are left unchanged in Step 5. Since we allocate  $\lceil \log_2(n_1 + 1) \rceil$  bits for each row pointer and  $\lceil \log_2(n_2 + 1) \rceil$  bits for each transition location, we can guarantee that the actual bit representation of each pointer or location will contain at least one transition.)

Steps 7 and 8 eliminate the  $t$ -bad columns using the following principle. Given any contents of  $\Gamma$ , denote by  $\Gamma(i)$  the  $(i - 1) \times n_2$  subarray defined by rows 2 through  $i$  in  $\Gamma$ . Also, let  $\bar{\Gamma}(i)$  be the array  $\Gamma(i)$  with its last row complemented (i.e., every '0' is changed into '1' and vice versa). Let  $\tau$  be the smallest integer such that  $\Gamma(i - 1)$  is not  $\tau$ -conservative. Now, for  $i > 2$ , the numbers of  $\tau$ -bad columns in  $\Gamma(i)$  and  $\bar{\Gamma}(i)$  sum up to the number of  $\tau$ -bad columns in  $\Gamma(i - 1)$ . Hence, in one of the arrays  $\Gamma(i)$  or  $\bar{\Gamma}(i)$ , the number of  $\tau$ -bad columns is at most one half that number in  $\Gamma(i - 1)$ . Therefore, we can eliminate all the  $\tau$ -bad columns in  $\Gamma(n_1)$  by complementing up to  $\lceil \log_2(n_2 + 1) \rceil$  rows in  $\Gamma$ . Doing this for

$\tau = 1, 2, \dots, t$ , we can eliminate all the  $t$ -bad columns from  $\Gamma$  by complementing up to  $K = t \lceil \log_2(n_2 + 1) \rceil$  rows. In our algorithm, we have chosen (arbitrarily) the last  $K$  rows in  $\Gamma$  to be considered for such complementation. We record in  $\mathbf{u}$  which rows are actually complemented (Step 8). Note that since  $\Gamma(n_1)$  does not contain any  $t$ -bad columns, neither will  $\Gamma$ , regardless of what we write in  $\mathbf{u}$ .

Counting the maximum number of bits that might be recorded in  $\mathbf{u}$  throughout the algorithm, we deduce that  $\mathbf{u}$  is large enough whenever condition (4) is satisfied. In particular, when  $t = 1$  and  $n_1 = n_2$ , it suffices to have  $n_1 \geq 12$  to achieve a redundancy of one bit. (In fact, we can slightly modify CONSERVATIVE to accommodate also the case  $n_1 = n_2 = 11$ ; observe that we need to leave one bit unchanged in  $\mathbf{u}$  in Step 5 only when  $\ell = 1$ , i.e., when no swap has been made in Step 4.) Encoders can be obtained also for smaller orders, in which case we will need to extend  $\mathbf{u}$  by additional redundancy bits.

The algorithm CONSERVATIVE can be implemented using  $O(n_1 n_2)$  bit operations and  $O(n_1 n_2)$  increments of  $\lceil \log_2 n_1 \rceil$ -bit counters.

Decoding is done as follows. If the upper-leftmost bit of the received conservative array  $\Gamma$  is '0' then the output array is also the input array. Otherwise, we effectively undo the encoding steps in reverse order, starting with complementing rows according to the  $K$ -bit subfield recorded in  $\mathbf{u}$  in Step 8. Next, we unfold the linked list and restore the  $t$ -bad rows. Finally, we reconstruct the original contents of the field  $\mathbf{u}$  by undoing the swap of Step 4.

Condition (4) can be relaxed at the expense of slightly increasing the encoding complexity, by incorporating a variation of the modulation technique of [27] in Step 7 of CONSERVATIVE. Specifically, let  $\oplus$  denote addition modulo 2, and let  $\mathcal{M}$  be a set of  $n_2 + 1$  binary words of length  $L$  such that the set of differential words

$$\partial\mathcal{M} = \{(v_1, v_1 \oplus v_2, v_2 \oplus v_3, \dots, v_{L-1} \oplus v_L) : (v_1, v_2, \dots, v_L) \in \mathcal{M}\}$$

is a  $(t - 1)$ -error-correcting binary code of length  $L$  and size  $n_2 + 1$ . As follows from [27], for every binary  $(L + 1) \times n_2$  array  $A$ , there is at least one element  $\mathbf{v} \in \mathcal{M}$  such that when  $\mathbf{v}$  is added modulo 2 to the  $L$ -suffix of each column of  $A$ , an  $(L + 1) \times n_2$  array  $A'$  is obtained in which every column is  $t$ -good. On the other hand, the number of transitions in each row of  $A$  is preserved in  $A'$ .

We can take  $\mathcal{M}$  such that  $\partial\mathcal{M}$  is a subset of a shortened binary  $(t-1)$ -error-correcting BCH code of length  $L$  and dimension  $\lceil \log_2(n_2+1) \rceil$  [13, pp. 258, 592]. Such a code exists whenever

$$L \geq \lceil \log_2(n_2+1) \rceil + (t-1)\lceil \log_2(L+1) \rceil. \quad (5)$$

If, in addition, we have  $n_1 \geq L+2$ , we can replace Step 7 and use such a set  $\mathcal{M}$  to eliminate the  $t$ -bad columns in  $\Gamma$  by adding an appropriate element  $v \in \mathcal{M}$  to the  $L$ -suffixes of the columns of  $\Gamma$ . In this case, it will be sufficient to record in Step 8 only the  $K = \lceil \log_2(n_2+1) \rceil$  bits that identify such an element  $v$ , thus allowing us to relax condition (4) to

$$n_1 \geq n_2 \geq 3 + \lceil \log_2(n_1+n_2) \rceil + t\lceil \log_2(n_2+1) \rceil \quad (6)$$

[compare with (2)]. Note that (6) implies (5) if we choose  $L = n_2 - 2$ , in which case we also have  $n_1 \geq L + 2$ . A brute force search for the appropriate word  $v \in \mathcal{M}$  to apply to  $\Gamma$  requires  $Ln_2^2$  additions modulo 2. Hence, it is preferable to have the smallest  $L$  that satisfies (5).

If we want to incorporate the weakly balancing property to  $t$ -conservative arrays, we can use the technique of [27, Sec. III.B], which is applicable also here.

One possible extension of the algorithm CONSERVATIVE is to the encoding of  $t \times t$  super-arrays in which each entry is a conservative  $n_1 \times n_2$  array. This way we obtain  $t$ -conservative  $tn_1 \times tn_2$  arrays in which the longest vertical (respectively, horizontal) run of identical bits does not exceed  $2n_1 - 2$  (respectively,  $2n_2 - 2$ ). It turns out that the same redundancy bit can be shared by all  $t^2$  arrays provided that  $n_1, n_2$ , and  $t$  satisfy an inequality analogous to (4). This is done, first, by swapping the upper-leftmost array (excluding its upper-leftmost bit) with a nonconservative array (if one exists), and using the upper-leftmost bit of that array to indicate whether all the arrays were initially conservative. Then, the nonconservative arrays are made into a linked-list, starting in (the field  $\mathbf{u}$  of) the upper-leftmost array. The changes made in each nonconservative array now follow along similar lines as those in CONSERVATIVE for  $t = 1$ , except that there is no need to allocate any redundancy bits within each nonconservative array.

We also mention that our algorithm can be generalized to encoding  $d$ -dimensional  $t$ -conservative  $n_1 \times n_2 \times \dots \times n_d$  hyper-arrays, where  $d \geq 2$  and  $n_1 \geq n_2 \geq \dots \geq n_d$ . The entry indexed by  $(1, 1, \dots, 1)$  is allocated for the redundancy bit, and the word  $\mathbf{u}$  occupies the entries that are indexed by  $(1, 1, \dots, 1, i)$ ,  $i = 2, 3, \dots, n_d$ . By an  $r$ -column in the hyper-array we refer to a set of entries in the hyper-array indexed by  $(u_1, u_2, \dots, u_{r-1}, i, u_{r+1}, \dots, u_d)$ , where the  $u_r$ 's are fixed and  $i$  ranges between 1 and  $n_i$ . The  $t$ -bad  $d$ -columns in the hyper-array are formed into a linked list, similar to the one in Step 5 of CONSERVATIVE. Then, for each  $r < d$  we remove the  $t$ -bad  $r$ -columns as follows. Define  $m_r = \prod_{j \neq r} n_j$  and  $K_r = t\lceil \log_2(m_r+1) \rceil$ , and let  $\Gamma(i, r)$  be the  $(i-1) \times m_r$  array whose columns are given by entries 2 through  $i$  of the  $r$ -columns. For  $n_r - K_r < i \leq n_r$ , we complement the  $m_r$  entries that are indexed by the  $(d-1)$ -dimensional hyper-array

$$\{(u_1, u_2, \dots, u_{r-1}, i, u_{r+1}, \dots, u_d) : 1 \leq u_j < n_j\}$$

if such complementation makes  $\Gamma(i, r)$  more conservative. Finally, using  $\sum_{r=1}^{d-1} K_r$  bits in the field  $\mathbf{u}$ , we record the hyper-planes that were complemented. (Alternatively, the  $t$ -bad  $r$ -columns for  $r < d$  can be handled using the error-correcting code approach which we discussed earlier.)

### III. ENCODING TWO-DIMENSIONAL DC-FREE ARRAYS

We present here an encoder that maps unconstrained binary words into binary  $n_1 \times n_2$  arrays in which each row and each column is

balanced. To this end, both  $n_1$  and  $n_2$  must be even. For the sake of simplicity we will further assume in our description that  $n_2$  is a power of 2. We will point out later on how the encoder can be adapted to handle any even value of  $n_2$ .

Our encoder is presented through the recursive algorithm BALANCE in Fig. 2. The procedure BALANCE first balances each row in the array, using any of the known algorithms for word balancing; see Knuth [12], Al-Bassam and Bose [1], Tallini *et al.* [26], and the enumerative coding technique in [24, p. 117]. The balancing of the rows results in an  $m \times n_2$  array  $\tilde{\Gamma}$ , where  $n_2$  is the desired final number of columns (Step 1). We point out that even though we assume that  $n_1 \geq n_2$ , we may apply Step 1 with values of  $m$  that are smaller than  $n_2$ . In particular, we take care of very small values of  $m$  separately (Step 2), where the balancing of the columns is carried out simply by appending a copy of the complement of  $\tilde{\Gamma}$  to  $\tilde{\Gamma}$ . For most values of  $m$ , we proceed with Steps 3–6 in BALANCE.

Next, BALANCE invokes a recursive procedure called SWAP which balances the columns by swapping pairs of bits in  $\tilde{\Gamma}$  as we now describe. We say that an array is *weakly balanced* if it contains the same number of '0's and '1's (namely, if it is balanced when regarded as a word). We set an  $m \times k$  array  $A$  to be initially  $\tilde{\Gamma}$  (Step 3 in BALANCE) and subdivide  $A$  into two disjoint subarrays  $A_1$  and  $A_2$ , each of order  $m \times (k/2)$  (Step a in SWAP). Note that  $A$  is weakly balanced, and this property will be preserved throughout the recursion levels of SWAP.

Step b in SWAP is essentially an application of the balancing technique of Knuth [12]. First, we index the bits in each of the subarrays,  $A_1$  and  $A_2$ , by integers between 1 and  $mk/2$ . For increasing values of  $i = 1, 2, \dots$ , we swap bit  $i$  in  $A_1$  with its counterpart in  $A_2$  until  $A_1$  becomes weakly balanced. Since  $A$  is weakly balanced, such a balancing point always exists (see [12]); indeed, if the number of '1's in  $A_1$  is greater than  $mk/4$ , then, clearly, the number of '1's in  $A_2$  is smaller than  $mk/4$ . Each bit swap changes the number of '1's in  $A_1$  by  $\pm 1$ , and if we swap  $A_2$  entirely with  $A_1$ , then the number of '1's in  $A_1$  drops below  $mk/4$ . Hence, there must be a bit swap for which the number of '1's in  $A_1$ , and in  $A_2$ , becomes exactly  $mk/4$ . We denote the first index  $i$  for which this occurs by  $\ell(A)$  (Step c in SWAP). After balancing  $A_1$  and  $A_2$ , we apply SWAP recursively to each of the subarrays  $A_1$  and  $A_2$  (Step d in SWAP). The recursion ends when  $k = 2$ , in which case both  $A_1$  and  $A_2$  are  $m \times 1$  balanced columns.

Consider a recursive execution of SWAP that is initiated by the call in Step 3 of BALANCE. It is easy to see that for each  $k = n_2, n_2/2, n_2/4, \dots, 2$ , the number of times that SWAP is called with an  $m \times k$  input array  $A$  is  $n_2/k$ , and in each such call we need  $\lceil \log_2(mk/2) \rceil$  bits to record  $\ell(A)$ . The total number of calls to SWAP is therefore  $n_2 - 1$ , and the total number of bits that we need in order to store all the values  $\ell(A)$  that are computed throughout the recursion levels of SWAP is

$$\begin{aligned} \sum_{\substack{k=2^j \\ 2 \leq k \leq n_2}} (n_2/k) \lceil \log_2(mk/2) \rceil &= \sum_{j=1}^{\log_2 n_2} (n_2/2^j)(j-1 + \lceil \log_2 m \rceil) \\ &= (n_2 - 1)(1 + \lceil \log_2 m \rceil) - \log_2 n_2. \end{aligned} \quad (7)$$

The call to SWAP in Step 3 of BALANCE transforms  $\tilde{\Gamma}$  into an array in which all the rows and columns are balanced. However, we still need to code the values  $\ell(A)$  that were computed in SWAP so that the decoding side will be able to reconstruct the contents of the array  $\tilde{\Gamma}$  before the call to SWAP. To this end, the values  $\ell(A)$  are concatenated in Step 4 into a word  $\mathbf{s}'$  whose length is as in (7). The word  $\mathbf{s}'$  undergoes the encoding process, recursively, to produce an

---

**procedure** BALANCE(input: integer  $n_2$ , word  $\mathbf{s}$ ; output: integer  $n_1$ , array  $\Gamma$  of order  $n_1 \times n_2$ )

/\* The number,  $n_1$ , of rows in  $\Gamma$  will be determined by the length of  $\mathbf{s}$ . \*/

1. For the smallest possible even  $m$ , encode  $\mathbf{s}$  into an  $m \times n_2$  array  $\tilde{\Gamma}$  in which every row is balanced.
2. If  $\tilde{\Gamma}$  has 12 rows or less then append the complement rows to generate an output array  $\Gamma$  and **return**. Otherwise proceed with Steps 3–6.
3. SWAP( $m, n_2, \tilde{\Gamma}$ ).
4. Concatenate the values  $\ell(\cdot)$  that were computed in all applications of Step c of SWAP (throughout the recursions levels), into a word  $\mathbf{s}'$  of  $(n_2-1)(1 + \lceil \log_2 m \rceil) - \log_2 n_2$  bits.
5. BALANCE( $n_2, \mathbf{s}', m', \Gamma'$ ).
6. Append  $\Gamma'$  to  $\tilde{\Gamma}$ , resulting in an output array  $\Gamma$  with  $n_1 = m + m'$  rows and  $n_2$  columns.

**procedure** SWAP(input: integers  $m, k$ ; input and output:  $m \times k$  array  $A$ )

- a. Let  $A_1$  be an array that consists of the first  $k/2$  columns of  $A$  and let  $A_2$  be the remaining half.
- b. For increasing values of  $i$ , swap the  $i$ th bit in  $A_1$  (according to some standard scanning) with the  $i$ th bit in  $A_2$ , until both  $A_1$  and  $A_2$  become weakly-balanced.
- c. Let  $\ell(A)$  be the final value of  $i$  that was reached in Step b (or  $\ell(A) \leftarrow 0$  if no swaps were made).
- d. If  $k > 2$  then do SWAP( $m, k/2, A_1$ ) and SWAP( $m, k/2, A_2$ ).

---

Fig. 2. Encoding algorithm for 2-D DC-free arrays.

$m' \times n_2$  array  $\Gamma'$  in which all the rows and columns are balanced (Step 5). The array  $\Gamma'$ , in turn, is appended to  $\tilde{\Gamma}$  (Step 6).

The resulting encoded array has order  $n_1 \times n_2$  where  $n_1 = m + m'$ . We next compute the redundancy of the encoder. Denote by  $\rho(m \times n_2)$  the redundancy that corresponds to the case where Step 1 produces  $m$  rows. The redundancy introduced by Step 1 is  $m(\frac{1}{2} \log_2 n_2 + O(1))$  if enumerative coding is used for balancing the rows [24, p. 117] and approximately twice that redundancy if any of the algorithms in [1], [12], and [26] is used. The redundancy introduced by Step 4 is less than  $n_2(1 + \lceil \log_2 m \rceil)$ . When applying Step 5 to  $\mathbf{s}'$ , the row balancing (Step 1 of the second recursion level of BALANCE) will clearly result in an array with ( $n_2$  columns and) no more than  $2(1 + \lceil \log_2 m \rceil)$  rows. Thus

$$\rho(m \times n_2) \leq m \left( \frac{1}{2} \log_2 n_2 + O(1) \right) + n_2(1 + \lceil \log_2 m \rceil) + \rho(2(1 + \lceil \log_2 m \rceil) \times n_2).$$

This recurrence readily implies the inequality

$$\rho(m \times n_2) \leq \frac{1}{2} m \log_2 n_2 + n_2 \log_2 m + O(m + n_2 + \log m \log n_2 + n_2 \log \log m).$$

It follows that  $n_1 = m + \log_2 m + O(\log \log m)$ ; thus, the redundancy of our encoder is at most

$$\frac{1}{2} n_1 \log_2 n_2 + n_2 \log_2 n_1 + O(n_1 + n_2 \log \log n_1)$$

assuming that  $n_1 \geq n_2$  and that enumerative coding is used for balancing the rows in Step 1 [compare with (3)]. If any of the algorithms in [1], [12], and [26] is used, then the redundancy increases by an additive term  $\frac{1}{2} n_1 \log_2 n_2$ . Note that the claimed redundancy holds even when we increase the threshold for  $m$  in Step 2 so that there is only one recursion call to BALANCE in Step 5 (the current threshold 12 was set so that the value of  $m$  will strictly decrease in each recursive call to BALANCE, regardless of the value of  $n_2$ ).

As for the time complexity of the encoding, Step 1 requires word balancing of  $O(n_1)$  rows, and the recursive calls to SWAP that are initiated in Step 3 require  $O(n_1 n_2 \log n_2)$  bit swaps.

Finally, we point out the changes that need to be made in SWAP when  $n_2$  is not a power of 2. Here, the value of  $k$  can be odd, in which case we define  $A_1$  to consist of the first  $(k-1)/2$  columns of  $A$ , whereas  $A_2$  consists of the remaining  $(k+1)/2$  columns. If  $A_1$  is already weakly balanced, no swaps are required in Step b of SWAP. Otherwise, we denote by  $A'_2$  an  $m \times (k-1)/2$  subarray of  $A_2$  whose imbalance is at a different direction than that of  $A_1$ ; e.g., if the number of '1's in  $A_1$  is greater than  $m(k-1)/2$ , then we require that the number of '1's in  $A'_2$  be smaller than  $m(k-1)/2$ . Such a subarray  $A'_2$  can be obtained by excluding from  $A_2$  a column, if any, in which the imbalance is at the same direction as that of  $A_1$ . The swaps in Step b are now carried out between  $A_1$  and  $A'_2$ , and the index of the excluded column is appended to  $\ell(A)$  in Step c. The respective recursive calls in Step d take the form SWAP( $m, (k-1)/2, A_1$ ) and SWAP( $m, (k+1)/2, A_2$ ).

#### APPENDIX

We verify here the bound (1). Let  $B(n, t)$  denote the number of  $t$ -bad words of length  $n$ . It is easy to see that

$$B(n, t) = 2 \sum_{i=0}^{t-1} \binom{n-1}{i} \leq 2n^{t-1}.$$

(This, in turn, implies  $\log_2 B(n, t) \leq 1 + (t-1) \log_2 n$ , which means that  $t$ -bad words of length  $n$  do not carry more than  $1 + (t-1) \log_2 n$  bits of information.) The number of binary  $n_1 \times n_2$  arrays with at least one  $t$ -bad row does not exceed  $n_1 B(n_2, t) 2^{n_1 n_2 - n_2}$  and, similarly, the number of binary  $n_1 \times n_2$  arrays with at least one  $t$ -bad column does not exceed  $n_2 B(n_1, t) \cdot 2^{n_1 n_2 - n_1}$ . Hence

$$\begin{aligned} |\mathcal{C}(n_1 \times n_2, t)| &\geq 2^{n_1 n_2} - n_1 B(n_2, t) \cdot 2^{n_1 n_2 - n_2} - n_2 B(n_1, t) \cdot 2^{n_1 n_2 - n_1} \\ &= 2^{n_1 n_2} (1 - n_1 B(n_2, t) \cdot 2^{-n_2} - n_2 B(n_1, t) \cdot 2^{-n_1}) \\ &\geq 2^{n_1 n_2} (1 - 2n_1 B(n_2, t) \cdot 2^{-n_2}) \end{aligned}$$

where the last inequality follows from our assumption that  $n_1 \geq n_2$ ; indeed, the probability,  $B(n_2, t) \cdot 2^{-n_2}$ , of getting less than  $t$  "heads"

when flipping an even coin  $n_2 - 1$  times is at least the probability,  $B(n_1, t) \cdot 2^{-n_1}$ , of that event when flipping the coin  $n_1 - 1$  times.

Taking logarithms, we obtain

$$\begin{aligned} \rho(C(n_1 \times n_2, t)) &\leq -\log_2(1 - 2n_1 B(n_2, t) \cdot 2^{-n_2}) \\ &\leq -\log_2(1 - 4n_1 \cdot n_2^{t-1} \cdot 2^{-n_2}) \\ &= -\log_2(1 - 4 \cdot 2^{-\Delta(n_1 \times n_2, t)}) \end{aligned}$$

thus proving (1).

#### REFERENCES

- [1] S. Al-Bassam and B. Bose, "On balanced codes," *IEEE Trans. Inform. Theory*, vol. 36, pp. 406–408, 1990.
- [2] J. J. Ashley, M. Blaum, and B. H. Marcus, "Report on coding techniques for holographic storage," IBM Res. Rep. RJ 10013, 1996.
- [3] J. J. Ashley and B. H. Marcus, "Two-dimensional low-pass filtering codes," *IEEE Trans. Commun.*, vol. 46, pp. 724–727, 1998.
- [4] R. Berger, "The undecidability of the Domino problem," *Mem. Amer. Math. Soc.*, vol. 66, 1966.
- [5] D. Brady and D. Psaltis, "Control of volume holograms," *J. Opt. Soc. Am. A*, vol. 9, pp. 1167–1182, 1992.
- [6] N. Calkin and H. S. Wilf, "The number of independent sets in a grid graph," *SIAM J. Discrete Math.*, vol. 11, pp. 54–60, 1997.
- [7] T. Etzion, "Cascading methods for runlength-limited arrays," *IEEE Trans. Inform. Theory*, vol. 43, pp. 319–324, 1997.
- [8] T. Etzion and V. K. Wei, "On two-dimensional runlength-limited codes," *IEEE Int. Workshop on Information Theory*, Salvador, Brazil, 1991.
- [9] S. Forchhammer and J. Justesen, "Entropy bounds for constrained two-dimensional random fields," *IEEE Trans. Inform. Theory*, vol. 45, pp. 118–127, 1999.
- [10] J. F. Heanue, M. C. Bashaw, and L. Hesselink, "Volume holographic storage and retrieval of digital data," *Science*, vol. 265, pp. 749–752, 1994.
- [11] —, "Channel codes for digital holographic data storage," *J. Opt. Soc. Am. A*, vol. 12, 1995.
- [12] D. E. Knuth, "Efficient balanced codes," *IEEE Trans. Inform. Theory*, vol. 32, pp. 51–53, 1986.
- [13] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: North-Holland, 1977.
- [14] M. Marcellin and H. Weber, "Two-dimensional modulation codes," *IEEE J. Selected Areas Commun.*, vol. 10, pp. 254–266, 1992.
- [15] B. H. Marcus, R. M. Roth, and P. H. Siegel, "Constrained systems and coding for recording channels," in *Handbook of Coding Theory*, V. S. Pless and W. C. Huffman, Eds. Amsterdam, The Netherlands: Elsevier, pp. 1635–1764.
- [16] B. H. Marcus, P. H. Siegel, and J. K. Wolf, "Finite-state modulation codes for data storage," *IEEE J. Select. Areas Comm.*, vol. 10, pp. 5–37, 1992.
- [17] E. K. Orcutt and M. W. Marcellin, "Enumerable multi-track  $(d, k)$  block codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 1738–1744, 1993.
- [18] —, "Redundant multi-track  $(d, k)$  codes," *IEEE Trans. Inform. Theory*, vol. 39, pp. 1744–1750, 1993.
- [19] P. Pavlidis, J. Swartz, and Y. P. Wang, "Fundamentals of bar code information theory," *Computers*, vol. 23, pp. 74–86, 1990.
- [20] D. Psaltis and F. Mok, *Holographic Memories*, *Scientific American*, vol. 273, no. 5, pp. 70–76, 1995.
- [21] E. Ordentlich and R. M. Roth, "On the redundancy of two-dimensional balanced codes," Hewlett-Packard Lab. Tech. Rep. HPL-97-143, 1997; also in *Proc. Int. Symp. Information Theory*, Cambridge, U.K., Aug. 1998, p. 113.
- [22] D. Psaltis, M. A. Neifeld, A. Yamamura, and S. Kobayashi, "Optical memory disks in optical information processing," *Appl. Optics*, vol. 29, pp. 2038–2057, 1990.
- [23] R. M. Robinson, "Undecidability and nonperiodicity for tilings of the plane," *Inventiones Math.*, vol. 12, pp. 177–209, 1971.
- [24] K. A. S. Immink, *Coding Techniques for Digital Recorders*. New York: Prentice-Hall, 1991.
- [25] G. T. Sincerbox, "Holographic storage re-visited," *Current Trends in Optics*, C. Dainty, Ed. New York: Academic, 1994.
- [26] L. G. Tallini, R. M. Capocelli, and B. Bose, "Design of some new balanced codes," *IEEE Trans. Inform. Theory*, vol. 42, pp. 790–802, 1996.
- [27] A. Vardy, M. Blaum, P. H. Siegel, and G. T. Sincerbox, "Conservative arrays: Multidimensional modulation codes for holographic recording," *IEEE Trans. Inform. Theory*, vol. 42, pp. 227–230, 1996.
- [28] H. S. Wilf, "The problem of the kings," *Electronic J. Combinatorics*, vol. 2, no. R3, 1995.

## An Improved Upper Bound of the Rate of Euclidean Superimposed Codes

Zoltán Füredi and Miklós Ruzsínkó

**Abstract**—A family of  $n$ -dimensional unit norm vectors is an *Euclidean superimposed code* if the sums of any two distinct at most  $m$ -tuples of vectors are separated by a certain minimum Euclidean distance  $d$ . Ericson and Györfi [8] proved that the rate of such a code is between  $(\log m)/4m$  and  $(\log m)/m$  for  $m$  large enough. In this paper—improving the above long-standing best upper bound for the rate—it is shown that the rate is always at most  $(\log m)/2m$ , i.e., the size of a possible superimposed code is at most the root of the size given in [8]. We also generalize these codes to other normed vector spaces.

**Index Terms**—Codes, growth rate, superimposed geometric codes.

### I. SUPERIMPOSED CODES

Binary superimposed codes were introduced by Kautz and Singleton [18]. They studied binary codewords with the property that the disjunctions (bitwise OR) of any pair of distinct at most  $m$ -tuples of codewords have to be different. Later, this question has been investigated in several papers on multiple-access communication (see, e.g., [1], [4], [5], [14]–[16]). The same problem has been posed—in different terms—by Erdős [6] and Frankl and Füredi [7] in combinatorics, by Sós [22] in combinatorial number theory, and by Hwang [11] and Sós [12] in group testing. One can find an easy proof of the best known upper bound of these codes in the papers by Füredi [9] and Ruzsínkó [21]. In the paper of Füredi and Ruzsínkó [10], the connection of these codes to the big distance ones is shown.

In 1988, Ericson and Györfi [8] introduced a new class of superimposed codes for Euclidean channels. Roughly speaking, a family of  $n$ -dimensional unit norm vectors is an *Euclidean superimposed code*, if the sums of any two distinct at most  $m$ -tuples of vectors are separated by a certain minimum Euclidean distance  $d$ . Similarly

Manuscript received December 17, 1997; revised August 25, 1998. This work was supported in part by the Hungarian National Science Foundation under Grant OTKA 016389, National Security Agency under Grant MDA904-98-1-0022, and OTKA under Grants T 016414 and F 014919.

Z. Füredi is with the Department of Mathematics, University of Illinois, Urbana, IL 61801 USA (e-mail: z-furedi@math.uiuc.edu) and the Mathematical Institute of the Hungarian Academy of Sciences, Budapest 1364, Hungary (e-mail: furedi@math-inst.hu).

M. Ruzsínkó is with the Department of Mathematical Sciences, Carnegie-Mellon University, Pittsburgh, PA 15213 USA (e-mail: ruzsínko@andrew.cmu.edu) and the Computer and Automation Research Institute of the Hungarian Academy of Sciences, Budapest 1518, Hungary (e-mail: ruzsínko@lutra.sztaki.hu).

Communicated by A. Barg, Associate Editor for Coding Theory. Publisher Item Identifier S 0018-9448(99)01415-7.