A Recursive Approach to Low Complexity Codes

R. MICHAEL TANNER, MEMBER, IEEE

Abstract—A method is described for constructing long error-correcting codes from one or more shorter error-correcting codes, referred to as subcodes, and a bipartite graph. A graph is shown which specifies carefully chosen subsets of the digits of the new codes that must be codewords in one of the shorter subcodes. Lower bounds to the rate and the minimum distance of the new code are derived in terms of the parameters of the graph and the subcodes. Both the encoders and decoders proposed are shown to take advantage of the code's explicit decomposition into subcodes to decompose and simplify the associated computational processes. Bounds on the performance of two specific decoding algorithms are established, and the asymptotic growth of the complexity of decoding for two types of codes and decoders is analyzed. The proposed decoders are able to make effective use of probabilistic information supplied by the channel receiver, e.g., reliability information, without greatly increasing the number of computations required. It is shown that choosing a transmission order for the digits that is appropriate for the graph and the subcodes can give the code excellent burst-error correction abilities. The construction principles are illustrated by several examples.

I. Introduction

THE EXPLICIT use of recursion in the construction of error-correcting codes is not new. In addition to the myriad construction techniques shown in [1], many well-known codes involve construction of a long length code from shorter codes. For example, Elias' product codes are straightforwardly recursive [2]; the Reed–Muller codes can also be seen to be a recursive construction when viewed as modifications of the basic product code construction [3]; and similarly Forney's concatenated codes are another variation on the recursive theme of product codes, although there is no apparent advantage to more than one stage of recursion within his framework [4]. It is noteworthy that in all of these constructions the conceptual simplicity of the recursion goes hand in hand with a simplicity in the computational process required for their implementation. Although the threshold decoder for the Reed–Muller codes can tend to hide the recursive structure, in all three of these techniques the conceptual decomposition of the recursion leads to a corresponding simplifying decomposition of the computational process. Unfortunately, all three would likewise appear to testify that simplicity comes with a certain price; the decomposition that affords the simplicity appears to prevent the long versions of the codes from being of the quality that is known to be possible, and the simple decoding algorithms are not even able to take advantage of the full power of the code. While modifications to both the codes and the decoding algorithms can improve the situation (e.g., for product codes, [5] and [6]), the improvements increase the complexity.

Recursion has been a central concept in a closely related field of study: the design and analysis of computer algorithms [7] and complexity theory. The complexity of a particular problem can often be understood by showing how the problem can be decomposed into smaller problems of the same type. For example, an algorithm for sorting a large unordered list of numbers consists of breaking the list into smaller lists, sorting each of these, and then merging the small sorted lists into successively larger sorted lists. The analysis of the algorithm then focuses on the work required for the merging process, because it is the merge work needed for a given subdivision scheme that ultimately determines the growth of complexity. This same line of attack has proven fruitful in a wide variety of problems.

In this paper we introduce a recursive approach to the construction of codes which generalizes the product code construction and suggests that the design of algorithms for encoding and decoding is amenable to the basic techniques of complexity theory. Long codes are built from a bipartite graph and one or more subcodes; a new code is defined explicitly by its decomposition into shorter subcodes. These subcodes are then used by the decoder as centers of local partial computations that, when performed iteratively, correct the errors. The decoding algorithms we propose generalize and unify the decoding schemes originally presented by Elias for his product codes and those of Gallager's low-density parity-check codes [8, pp. 41–52]. Correspondingly, the asymptotic growth rate is also comparable to that of these two low-complexity schemes. This paper provides a proof of a relatively weak lower bound on the minimum distance of the block code constructions and a related decoding algorithm's correction abilities, but our study of short codes indicates that these low-complexity codes can have minimum distances as great as those of the best-known codes, and that simple algorithms can correct the errors effectively. Furthermore, the proper choice of the transmission order for the bits can guarantee good performance against burst errors or a mixture of burst and random errors. The decoding by iteration of a fairly simple basic operation makes the suggested decoders naturally adapted to parallel implementation with large-scale-integrated cir-

Manuscript received January 17, 1979; revised September 8, 1980. This work was supported in part under a grant from the Faculty Research Committee of the University of California, Santa Cruz, CA.

The author is with the Board of Computer and Information Sciences, University of California, Santa Cruz, CA, 95064.

0018-9448/81/0900-0533$00.75

cuit technology. Since the decoders can use soft decision information effectively, and because of their low computational complexity and parallelism can decode large blocks very quickly, these codes may well compete with current convolutional techniques in some applications.

II. Code Construction

A new code is formed from a bipartite graph and one or more codes of shorter length, which will be referred to as subcodes. A bipartite graph is one in which the nodes can be partitioned into two disjoint classes. An edge of the graph may connect a node of one class to a node of the other class, but there are no edges connecting nodes of the same class. To define the new code, all of the nodes of one class are associated with digits of the code; all the nodes of the other are associated with a subcode whose length is the same as the degree of the node. Each of the edges incident on a subcode node is assigned a unique position number between one and n, the length of the associated subcode. It is then natural to think of the digits to which each subcode node is connected as forming a word in that subcode, with each digit assigned the position in each subcode corresponding to the label of the connecting edge. This defines the words in the new code: a pattern of digits is a codeword if and only if the digits connected to each subcode form a word in the subcode. The code is thus specified by requiring that particular permutations of subsets of the digits be codewords in the subcodes. This is illustrated in Fig. 1.

Without further restriction, this definition is too broad to be meaningful. Given the parity-check matrix of an arbitrary linear code, it is easy to construct a corresponding bipartite graph by identifying each row with a subcode node, each column with a digit node, and creating an edge in the graph for every nonzero matrix entry. This mapping of a code to a graph also serves to point out that the graph corresponding to a code is not unique. Different matrices defining the same code lead to different graphs.

By placing restrictions on the form of bipartite graphs and on the subcodes, it is easy to move beyond this trivial interpretation. The central idea is to use the graph to structure the equations defining the code in a way that facilitates encoding and decoding. To start the exploration of the effect of different restrictions, we first consider graph interpretations of three well-known coding schemes —low-density parity check codes, Bose–Chaudhuri–Hocquenqhem (BCH) codes (see e.g., [9, pp. 269–309]), and product codes—and some related examples.

An appealing uniformity is obtained by requiring that the graph be regular. Let the degree of every digit node be a constant j and the degree of every subcode node a constant k. By letting each of the subcode nodes represent a simple parity check, the graph defines a low-density parity-check code. Note that if as the name implies j and k are very small compared to the number of digit nodes (the length of the code), the graph corresponds to a sparse parity-check matrix. Further restriction to the case $j = 2$ and binary digits will of course yield a circuit code, since each digit node then is connected to only two subcodes and can serve to identify a single edge between the two [9, pp. 136–138].

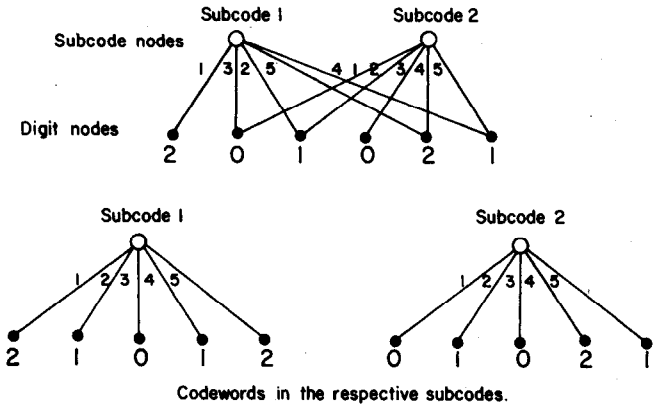

Fig. 1. Code definition.

When all of the subcodes are simple binary parity checks, there is obviously no need to assign position numbers to each of the bits in a subcode, as each of the subcodes is invariant under arbitrary permutation of its bits. Generally this is not the case, and the assignment of position numbers is of critical importance in determining both the rate and the minimum distance of the code. A straightforward graph representation of a BCH code will serve to demonstrate the potential difficulty of the assignment problem. A primitive binary BCH of prime length can be specified by requiring that all codeword polynomials have as roots several successive powers of α, where α is a primitive element of GF(2^m) and $2^m - 1$ is prime. For the purpose of defining a graph, each of the roots can be viewed as a subcode that checks all the bits. Since the codelength is prime, all of the subcodes are simply Hamming codes, but the position numbering is different for each root. Thus the codeword must be a word from a Hamming code under as many different permutations as there are nontrivial roots. Clearly in this case the entire question of code rate and minimum distance rests on understanding the effect of different position number assignments. The form of the graph alone gives little insight.

The impact of a natural graph representation is more evident in the case of the standard product code construction. A codeword of length $n_1 n_2$ is formed as an n_1 by n_2 rectangular array of digits where each row is a codeword of a code of length n_2, and each column is a codeword of a code of length n_1. To form a corresponding graph, create n_1 subcode nodes of degree n_2, one for each of the row constraints, and n_2 subcode nodes of degree n_1, one for each of the column constraints. The node corresponding to each digit is then connected to two subcode nodes, one for its row and one for its column. This is demonstrated in Figs. 2 and 3 using the product of two [7, 4, 3] Hamming codes.[1] The usual proof that the minimum distance of the

[1]Following the notation of [1], a linear code will be denoted by a 3-tuple, $[n, k, d_{\min}]$.

1	0	1	1	0	0	0
0	0	0	0	0	0	0
1	0	1	1	0	0	0
1	0	1	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Fig. 2. Standard [49, 16, 9] product code from [7, 4, 3] Hamming code.

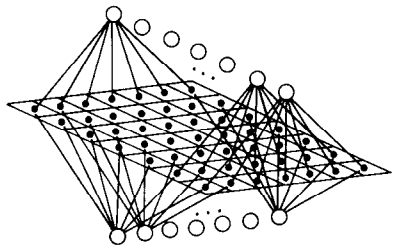

Fig. 3. Associated bipartite graph.

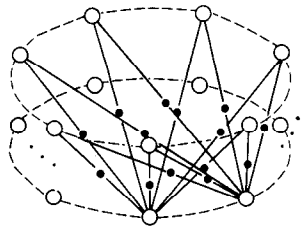

Fig. 4. The bipartite graph as two rings of seven subcode nodes.

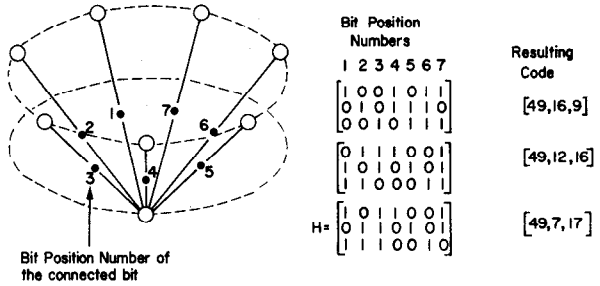

Fig. 5. The effect of different bit position assignments.

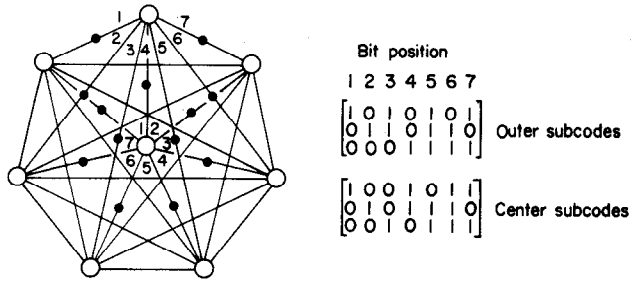

Fig. 6. A [28, 9, 10] code.

product code must be at least as great as the product of the minimum distances of the two constituent codes may now be rephrased in graph terms. Suppose two codewords differ in at least one digit. That digit is connected to one column subcode node, which implies because that subcode has minimum distance d_1, that the two codewords differ in at least d_1 of the digits connected to that subcode node. These in turn are connected to at least d_1 different row subcode nodes; each of these has minimum distance d_2 and no two of them are connected to the same bit. Thus the codewords must differ in at least $d_1 d_2$ digits. The point that we wish to emphasize is that this is a consequence of the properties of the graph. This lower bound on minimum distance holds independently of the particular position assignments of the digits in each subcode. The product code construction of course specifies position assignments for all of the digits, but the lower bound on the distance would be valid for any of the myriad possible assignments.

To demonstrate the effect of different assignments, consider once again the example of the product of two [7, 4, 3] binary Hamming codes. Starting from the bipartite graph of Fig. 3, we arrange it so that it appears as two rings of seven subcode nodes with a bit node between every pair of top and bottom nodes as shown in Fig. 4, and impose the condition that the subcode position assignments have circular and reflexive symmetry so that rotating the graph about its axis or reflecting about the midplane does not change the relationship between the bits and the subcodes at any position. The code will then be specified by choosing one of the 7! possible position assignments for the seven bits connected to a particular subcode node. Many of these choices will yield equivalent codes, obviously. Fig. 5 gives three assignments yielding markedly different codes: the cyclic Hamming code assignment leads to the standard [49, 16, 9] code; the next gives a [49, 12, 16] code; the third gives a [49, 7, 17] code. A comparable best-known code has parameters [49, 12, 17] [1, p. 678]. Note further that the largest minimum distance far exceeds the lower bound of nine guaranteed by the graph properties alone. Moving one step further, let us now consider a simple example that starts from a different graph.

A complete graph on 8 nodes has 28 edges, one connecting every pair of nodes. From this we create a bipartite graph by replacing each edge with a bit node and two edges so that there is now a unique bit node connected to each pair of nodes, which can then serve as subcode nodes. Since the subcode nodes have degree seven, associate each with a [7, 4, 3] Hamming code. Of the many possible ways of assigning bit positions, those with some natural symmetry seem most appealing. The code shown in Fig. 6 is built with the graph in the form of a spoked wheel; the hub subcode is the cyclic Hamming code, and the rest of the assignments have circular symmetry. It produces a [28, 9, 10] code, matching the best-known code [1, p. 676].

As both the modified product codes and the standard BCH code demonstrate, the choice of position assignments in the subcodes can dramatically affect the parameters of the resulting code. This paper will focus, however, on those code properties that are derivable from the graph parameters and the subcode minimum distance. Although the bounds on the code rate and minimum distance that result are weak when compared for example to the comparable BCH bounds, the computational properties of the algorithms we will present could make these codes attractive nonetheless. Convolutional codes in current use are simi-

larly weak in terms of rate and minimum distance, but their computational properties make them of practical importance. Moreover the evidence provided by short codes suggests that it is the graph-based bounds that are weak, not the codes. Future work may well establish tighter bounds.

For the sake of the discussion of encoding and transmission order we need more complete knowledge of the code structure than is provided by graph parameters and subcode minimum distance alone. The following class of codes will be used to demonstrate several techniques.

Field Plane Hexagon Codes

The projective plane PG(2, q) [10, p. 6] has $(q^3 - 1)/(q - 1)$ lines and an equal number of points. We create a subcode node for each point P and one for each line L, and a bit node for each point-line pair (P, L). A bit node (P, L) is connected in the obvious way to subcode node P and subcode node L. The resulting graph is a generalized hexagon [11, pp. 300–305]. It can be presented as a double ring similar to Fig. 4 with all point subcode nodes in the top ring and all line subcode nodes in the bottom: let α be a primitive element of GF(q^3) represented by a vector $\boldsymbol{\alpha} \in \mathbb{F}^3$, $\mathbb{F} = \mathrm{GF}(q)$. A point of PG(2, q) is a one-space of $\mathbb{F}^3$, i.e., $\{\gamma\boldsymbol{\alpha}\ \gamma \in \mathbb{F}\}$. Consequently the distinct points of PG(2, q) can be cyclically indexed by α^i, $i = 0, 1, \cdots, t-1$, with $t = ((q^3 - 1)/(q - 1))$ (α^t is a primitive element of $\mathbb{F}$) to form the top ring. Similarly, let $\boldsymbol{\beta} \in \mathbb{F}^3$ correspond to a particular line in PG(2, q) and let A be the 3×3 matrix over $\mathbb{F}$ that is the companion of α; that is, $A\gamma = (\alpha\gamma)$ for some $\gamma \in \mathrm{GF}(q^3)$. The line in the bottom ring below the point α^i is then $B^i\boldsymbol{\beta}$, $B = (A^T)^{-1}$. Incidence in PG(2, q) can be defined via the dot product operation in $\mathbb{F}^3$; namely, point $\boldsymbol{\alpha}_o$ is incident on line $\boldsymbol{\beta}_o$ whenever $\boldsymbol{\alpha}_o^T \cdot \boldsymbol{\beta}_o = 0$. The double ring presentation is circularly symmetric because if $\boldsymbol{\alpha}_o^T \cdot \boldsymbol{\beta}_o = 0$, then $(A^i\boldsymbol{\alpha}_o)^T \cdot (B^i\boldsymbol{\beta}_o) = \boldsymbol{\alpha}_o^T (A^T)^i B^i \boldsymbol{\beta}_o = \boldsymbol{\alpha}_o^T \cdot \boldsymbol{\beta}_o = 0$. Since each subcode has $q + 1$ incident bits, this graph can be combined with any code of length $q + 1$ to produce a code of length $(q + 1)(q^3 - 1)/(q - 1)$.

III. Code Properties

The properties of the bipartite graph alone can be used to guarantee lower bounds for the code rate and minimum distance. Obtaining a bound on rate is easy if all of the subcodes are linear, implying linearity of the resulting code.

Theorem 1: Let R be the rate of a linear code constructed from a bipartite graph whose digit nodes have degree m and whose subcode nodes have degree n. If a single linear subcode with parameters (n, k) and rate $r = k/n$ is associated with each of the subcode nodes, then

$$R \geq 1 - (1 - r)m. \tag{3.1}$$

Proof: Suppose there are S subcode nodes in the graph. Then the code must have $(n/m)S$ digits, since each digit node is connected to m of the $(n)S$ edges in the graph. Each subcode node contributes $n - k$ equations to the parity-check matrix for a total of $(n - k)S$. These equations may not be linearly independent. Therefore

$$R \geq \frac{(n/m)S - (n-k)S}{(n/m)S}$$

$$= 1 - m\left(\frac{n-k}{n}\right) = 1 - m(1 - r).$$

As an illustration of this simple theorem, consider the codes shown in Fig. 5. The digit nodes of this graph have degree two, so that the rate of these codes must be greater than $1 - 2(1 - 4/7) = 1/7$, which is achieved by the [49, 7, 17] code.

A lower bound on the minimum distance of the code can be derived from the minimum distance of the subcode and the girth of the bipartite graph. Note that the girth of a bipartite graph must always be even.

Theorem 2—Tree Bound on Minimum Distance: Let d be the minimum Hamming distance of the single subcode acting at all subcode nodes, D be the minimum Hamming distance of the resulting code, m be the degree of the digit nodes, and g be the girth of the bipartite graph. Then

$$D \geq d\frac{[(d-1)(m-1)]^{(g-2)/4} - 1}{(d-1)(m-1) - 1} + \frac{d}{m}[(d-1)(m-1)]^{(g-2)/4}, \quad \text{for } g/2 \text{ odd}, \tag{3.2}$$

$$D \geq d\frac{[(d-1)(m-1)]^{g/4} - 1}{(d-1)(m-1) - 1}, \quad \text{for } g/2 \text{ even}. \tag{3.3}$$

Proof: Suppose two codewords differ in a digit connected to a particular subcode node. Arrange the graph in the form of a tree with that subcode as root, as shown in Fig. 7. The two codewords must differ in at least d digits at the highest digit level. The minimum distance of the subcode nodes at the next level down ensures that, descending to the next digit level, the codewords will differ along $d(d-1)(m-1)$ branches. If all of these edges do not connect to distinct digit nodes, it implies that there is a cycle of length six. If the graph has girth greater than six, this next level of digits must have at least $d[(d-1)(m-1)]$ digits where the codewords differ. The number of differing digits increases by a factor of $(d-1)(m-1)$ at each new digit level, until at least a depth equal to $g/2$. If $g/2$ is odd, the first level where differing branches may join to form a cycle is a digit level, and each of these digits may terminate m different branches. This leads to the factor of $1/m$ in the last term. Summing the differing digits over all the levels yields

$$D \geq d\sum_{j=0}^{(g-6)/4}[(d-1)(m-1)]^j + \frac{d}{m}[(d-1)(m-1)]^{(g-2)/4} \tag{3.4}$$

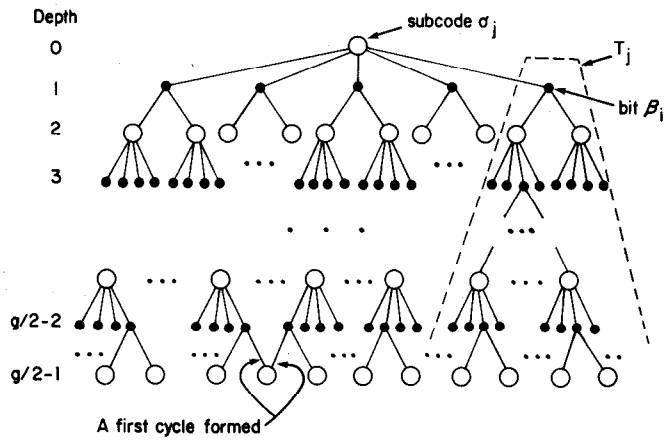

Fig. 7. Tree bound on minimum distance.

for $g/2$ odd and

$$D \geq d \sum_{j=0}^{(g-4)/4} [(d-1)(m-1)]^j \qquad (3.5)$$

for $g/2$ even, which reduce to the expressions given.

It is possible to derive a different bound by using a digit as the root, but the given bound will always be better so long as $d \geq m + 2$. It is interesting to note that this bound, which is closely related to arguments used by Gallager [8, pp. 18–20], reduces to the bound satisfied by the product code construction when the graph has girth eight.

Unfortunately the bound is relatively weak. While the distance D grows exponentially with the graph girth, so does the code length N. Looking again at the tree, it is easy to see that

$$N \geq n \frac{[(n-1)(m-1)]^{g/4} - 1}{(n-1)(m-1) - 1} \qquad (3.6)$$

for $g/2$ even. Consequently this bound alone does not prevent the ratio of minimum distance to code length from approaching zero as the girth of the graph is increased.

The bound of Theorem 2 will be strongest if the length of the code achieves equality in (3.6), thereby minimizing the length for a fixed lower bound on the minimum distance. Then all digits in the code are contained in the tree of Fig. 7 when it is extended to the level where the first cycle is formed. For such a graph, the graph diameter, which is the length of the path connecting the most distant pair of nodes in the graph, is $t = \lfloor g/2 \rfloor$. The girth of an undirected bipartite graph must always satisfy $g \leq 2t + 1$, as the following argument shows: let v_1 and v_2 be two nodes in the graph for which the shortest path has length t, and let v_3 be the first node encountered on another path from v_1 to v_2. By definition of the diameter, the minimum path from v_3 to v_2 must have length less than or equal to t. Clearly this path cannot include v_2. Therefore there is a cycle from v_1 to v_2 to v_3 and back to v_1 of length less than or equal to $2t + 1$. The condition $t = \lfloor g/2 \rfloor$ is therefore a fairly stringent requirement on the graph. With a graph satisfying this requirement, however, *any* node in the graph can be used as the root of the tree of Fig. 7 and the tree will contain all the nodes at depth $g/2 - 1$ or less.

Fortunately the existing literature provides many avenues for constructing regular graphs satisfying the girth condition approximately if not precisely. By inserting digit nodes in the edges of a Moore graph [12] or of a graph based on a generalized polygon [11, pp. 300–305], bipartite graphs with digit node degree two that satisfies the condition exactly can be obtained at a variety of sizes and girths. Although this mathematical approach cannot furnish nontrivial graphs with girths greater than 32 and $t = \lfloor g/2 \rfloor$, this poses no particular obstacle from a practical point of view. The field plane hexagon of Section II has a girth of only 12, yet can easily produce codes longer than current applications require.

An algorithmic attack on the graph construction problem is also possible. Using the matrix equivalent of the code graph, Gallager provided a construction to guarantee independent decoding iterations for his low-density parity-check codes [8, pp. 91–99]. The first $\lfloor g/2 - 1 \rfloor$ iterations are independent, and so his construction is precisely addressed to the issue of achieving a large girth for a given graph diameter. His construction will only ensure a girth approximately equal to the diameter, however. Given the exponential dependence of length on girth, his construction yields graphs much weaker than those of the generalized polygons.

IV. Encoding

The form of the best encoder for a code of this type will necessarily depend on the specific graph, subcodes, and position assignments. Nevertheless, fairly general considerations enable us to conclude that the complexity of encoding can be made at least comparable to the complexity of encoding normal cyclic codes, for example.

For linear codes, of course, the linearity alone guarantees a relatively simple algorithm; solving the defining equation $H \cdot C = 0$ for the parity check digits can be done in fewer than $K(N - K)$ multiplications and a like number of additions. The H matrix of a recursively defined code is sparse, and thus parallel algorithms for sparse matrices can be used to solve this equation [13]. With a cyclic code the null matrix has a simple representation, each row being a cyclic shift of a fixed row, which permits a single shift register with fixed connections to carry out the computations. The same strategy can be employed to good end with many of the recursively defined codes. What is required is that the code be invariant under a group of automorphisms sufficiently powerful to carry one codeword into an orbit which includes a basis for the code space. Encoding can then be implemented using registers which are wired to carry out the necessary transformations. Both generator matrix and null matrix versions of this concept will be discussed in the context of an example in Section VIII.

Another avenue to the implementation of encoders exploits the subcode constraints directly to reduce the required computation. The key idea is to use the subcode

constraints to propagate the solution to the constraint equations across the graph. The normal $[n^2, k^2, d^2]$ product code can be encoded in two steps using k subcode encodings followed by n subcode encodings. The following theorem shows a similar propagation algorithm that can be used in some of the field plane hexagon codes introduced in Section II.

Theorem 3: Given an $[n, k, d]$ subcode, there exists an assignment order in the nodes of a field plane hexagon such that knowledge of $k(k-1)n + k$ bits can be used to encode a codeword in five steps using fewer than $2((n-1)^3 - 1)/(n-2)$ subcode encodings.

Proof: Fig. 8 exhibits a field plane hexagon in tree form with the subcode nodes corresponding to lines in $PG(2, q)$ indicated by squares, those corresponding to points indicated by circles. The encoding can proceed in five steps.

1) An information set of k bits is specified in the top subcode and it is encoded.

2) An additional $(k-1)$ specified bits are chosen to complete the information set for each of the k line subcodes at depth one that share a specified bit with the top subcode, and they are encoded.

3) An additional $(k-1)$ specified bits are chosen to complete an information set for each of the $k(n-1)$ point subcodes that share a bit with the encoded subcodes of step 2), and they are encoded.

4) By the axioms for a projective plane satisfied by $PG(2, q)$, each of the line subcodes at the bottom of the tree is connected via a unique point subcode with each of the k line subcodes of step 2) at depth one. Thus each of the bottom subcodes has k incident bits that are either specified or computed in step 3). There exists an assignment order such that they form an information set in each, and the bottom subcodes can thus be encoded.

5) Each of $(n-k)(n-1)$ point subcodes at depth two then have $(n-1)$ incident bits computed in step 4) and the last remaining bit in each case can be encoded.

The total number of specified bits is, from the first three steps, $k + k(k-1) + k(n-1)(k-1) = k(k-1)n + k$. There are $2((n-1)^3 - 1)/(n-2)$ subcode nodes in the graph; all but $(n-k)$ at depth one are encoded at some stage, although for those encoded in step 5), only one bit is computed.

The need for a particular assignment order occurs only in the fourth step. If the k bits incident on a bottom subcode do not form an information set, it means that some of the specified bits could not have been freely chosen and that additional bits in that subcode may have to be specified for the encoding to be complete. For an arbitrary assignment, the set of known bits consisting of true information bits and possibly some generally computed bits, may have to be slightly larger.

This basic strategy can be adapted to a wide variety of graphs and subcodes. The subtlety lies in deducing from the graph and the specific properties of the subcodes the minimum amount of globally computed information needed to permit propagation of the solution by localized subcode computations.

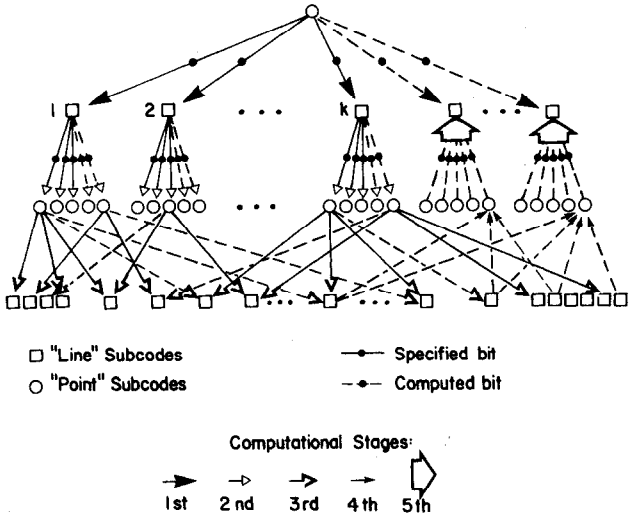

Fig. 8. Encoding a field plane hexagon code.

V. Transmission Order for the Digits

Thus far the digits of a recursively defined code have been treated only as corresponding to the unordered digit nodes of the graph, with the desirability of one order over another influenced simply by the encoder design. If the errors introduced by the channel tend to occur in bursts, however, the specific order of transmission becomes an important issue because it greatly affects the ability of the decoder to correct such burst errors.

Assuming that the code is based on subcodes with a certain minimum distance of their own, it is clear that the subcode structure alone provides for a large distance between any two codewords which differ in many subcodes. Consequently, an auspicious transmission order is one which causes errors occurring in bursts to appear as low weight errors in a large number of subcodes. Even if the subcodes themselves are designed to operate against purely random errors, following this strategy will enhance the burst error correction abilities of the new code. If, in addition, the subcodes themselves have superior burst error correction properties, the transmission order should make a burst of errors appear as many correctable bursts in the subcodes. This will be important for a code defined with multiple stages of recursion, since the subcodes of the last stage will be a smaller recursively defined code with guaranteed strength against burst errors. Since in general the order must be matched to both the graph and the subcode, the field plane hexagon construction will be used to illustrate the general strategy.

Theorem 4: A subcode of length n capable of correcting all bursts of length b or less can be used in a field plane hexagon to produce a code of length $n((n-1)^3 - 1)/(n-2)$ capable of correcting all bursts of length $b((n-1)^3 - 1)/(n-2)$ or less.

Proof: Let bits incident on the point subcode corresponding to $\alpha_o = 1$ be labeled arbitrarily zero to $n-1$ and those incident on any other be given the labeling induced by circular symmetry; that is, if the bit between α_o and β, $\boldsymbol{\alpha}_o^T\boldsymbol{\beta} = 0$, is labeled j, the edge between $A^i\boldsymbol{\alpha}_o$ and $B^i\boldsymbol{\beta}$ is likewise. A bit in the graph can then be indexed by (j, i), $j = 0, 1, \cdots, n-1$, and $i = 0, 1, \cdots, t-1$, $(t = (n-1)^3 - 1)/(n-2))$. Assign a bit position in each subcode according to label j. Transmit the bits in lexicographical order on (j, i), i.e., $(0,0)$, $(0,1), \cdots, (0, t-1)$, $(1,0), \cdots, (1, t-1), \cdots, (n-1, t-1)$. Obviously any burst of length $b((n-1)^3 - 1)/(n-2)$ or less will cause a burst of length b or less in each subcode. If each subcode corrects errors in its incident bits according to the subcode burst correction procedure, the entire burst will be corrected.

The graph in this case provides perfect interleaving for the point subcodes alone and for the line subcodes alone. The advantage of the graph structure is that the same memory is simultaneously organized to correct random errors as well.

An analogous order for a complete graph is given in Section VIII.

VI. Decoding

The principal objective of defining the codes in terms of explicit subcodes is to reduce the complexity of the decoding process to provide high quality codes that can be decoded effectively by a computational process whose complexity grows only very slowly with increasing code length at fixed code rate. Given the close relationship between our codes and both product codes and low-density parity-check codes, it is not surprising that the decoding algorithms we propose blend themes present in those earlier works. The crucial concept is the following: low complexity error-correction for the recursively defined code is achieved by the decomposition of the entire decoding process into an ensemble of independent partial decoding processes carried out, usually iteratively, by each of the constituent subcodes. Information is passed from one subcode to another only through statistics associated with a digit they have in common.

Before moving to the exposition of the more involved error-correction procedures, the use of a recursively defined code for detection warrants a brief comment. Since the codewords are defined to be any set of digits satisfying all of the subcode relations, the correctness of a codeword can be verified by simply verifying that all of the subcode constraints are satisfied. This in turn merely requires an error detection check for each of the subcodes. Thus detection for the new code is no more difficult than the sum of the detection operations for all of the defining subcodes.

The underlying concept of decomposing the decoding process can be embodied in a variety of algorithms. To give some idea of the range of possibilities, we will present three types of algorithms of increasing sophistication that incorporate aspects of product code decoders and probabilistic low-density parity-check decoders. It will be useful to think of them as being carried out by a parallel architecture: each subcode node in the graph has associated with it a special processor capable of executing a specific decoding procedure for the subcode at that node; each digit node has associated with it one or more registers accessible by the subcodes checking it and a processor for updating the contents of the digit's registers. With the exception of a central control that sequences and coordinates the executions at each node, the processors are autonomous and perform the computation solely on the contents of registers they access.

The first procedure generalizes the original product code decoding procedure. There is a single register for each digit which contains a current best estimate for that digit. Initially each register is loaded with the estimate for that digit supplied by the channel receiver. The central control then requests each subcode processor, one by one in some prescribed order, to attempt to correct any errors in the bits as perceived by that subcode. Each processor may change some digits, and the next subcode processor will operate on the new digits. The entire cycle of corrections may be repeated a fixed number of times or terminated earlier if no digits are changed.

In the second class of algorithms, there are multiple registers for each digit containing different estimates of the digit. There are two phases to the procedure. In the first, each subcode node processor receives a digit value from one of the registers of each of the digits it checks. Based on those values the subcode processor computes corrections to be made, and returns a "corrected" value for each digit to the corresponding digit register. In the second phase, the contents of all of the digit registers for each digit are updated by a computation based only on the contents of the registers for that digit. For example, this could be a threshold operation where all of the registers are set to the value occurring most frequently among the "corrected" values returned by the subcodes, or each register could be determined by a vote of all the other registers. The two phases are then alternated for a prescribed number of iterations. The final value for each digit is determined by a vote of all associated registers.

To illustrate with a specific algorithm of this type, consider a graph of girth g where all bit nodes have degree two, all subcodes have degree $2^m - 1$, and each subcode node has associated a binary Hamming code of length $2^m - 1$. There are three registers for each bit, one which stores the initial received value of the bit during the entire decoding process, and one for each of the two subcodes checking the bit.

Algorithm A: All three registers are initially loaded with the received value of the bit. During the first phase, each subcode takes in both a vector representing the initial values of all incident bits and a vector of the values currently in its own bit registers. It then calculates an updated vector of values to be returned to its registers as follows: the initial value of each bit is used along with the current values of all other bits to form a codeword; if the single error correction procedure for the Hamming code

indicates that the initial value is incorrect, then the updated value is the complement of the initial value; if the correction procedure indicates that either the word is correct or some other bit is in error, then the updated value is the initial value unchanged. This is carried out for every subcode in the code. Then for each bit the bit register values coming from the two subcodes are simply interchanged, so that the result of one subcode's correction is used in the next correction of the other. The two phases are alternated for $\lfloor (g-2)/4 \rfloor$ iterations. The final decoded value of a bit is the value in the two subcode registers when they agree, and the initial value when they disagree.

Algorithm A obviously unites principles from both product codes and low-density codes (see, specifically, [8, pp. 47–52]). It is probably not surprising then, that the probability of error in a bit can be made to go to zero asymptotically.

Theorem 5: Any code based on a bipartite graph with bit node degree two, subcode node degree $2^m - 1$, and the length $2^m - 1$ single-error correcting Hamming code can be decoded by Algorithm A with bit error probability approaching zero as the girth of the graph approaches infinity when used on a binary symmetric channel with crossover probability q if $q < 1/(2^m - 2)$.

Proof: Starting with an arbitrary bit as root, draw the code as a tree with that bit as root at the top. The nodes at even depths of the tree are other bits in the code. Looking first at the bits at depth $2\lfloor g/4 \rfloor$, the first subcode correction will cause the "corrected" bits at the next level up to have probability of error p_1. By Lemma 2 of Appendix A, $p_1 < rq$ for some $r < 1$. Clearly, these "corrected" bits are all statistically independent; consequently, after the next iteration the bits at depth $2\lfloor g/4 \rfloor - 2$ will have probability $p_2 < rp_1 < r^2 q$. Continued iterations give the "corrected" bit at the top of the tree a probability of error

$$p = r^{\lfloor g/4 \rfloor} q.$$

It follows immediately that p goes to zero as the girth goes to infinity.

Gallager's graph construction ensures the existence of graphs with arbitrarily large girths. Because the length of the code grows exponentially with the girth, this bit error convergence alone does not give asymptotically reliable communication. As the proof of the lemma reveals, however, by making the initial crossover probability q sufficiently small the ratio r can be made arbitrarily small, which will cause the probability of decoding error to go to zero. More rapid convergence of the decoding error probability can be obtained by using a more powerful subcode or a higher bit node degree.

One of the major sources of weaknesses of this type of algorithm is the loss of information that occurs during a subcode correction; the result passed to the next subcode is simply a "corrected" digit, with no indication of the reliability of that "corrected" digit. The third type of algorithm to be considered, which generalizes the probabilistic decoder of low-density parity-check codes, remedies this deficiency by using registers which carry probability estimates for the value of the associated digit.

For clarity of exposition we will present the algorithm for the binary case with digit degree three. At each bit node there is one register that stores a hard decision value for the received bit and another that stores the probability that value is wrong based on information from the receiver. In addition there is a separate probability register for each subcode that checks the bit. These registers are initially loaded with the probability of bit error from the channel. At the start of the process each subcode calculates a syndrome using the hard decision bit values. This syndrome is saved to be used throughout the process. As before, multiple iterations of a two-phase process carry statistically independent information across the graph. The computations which take place during a single iteration are shown in Fig. 9. The underlying theory is the following. Let p_i be the probability, supplied by the receiver, that the original hard decision for bit i is in error. Let p_{i1} be the updated probability that the bit is in error given three statistics: first, p_i; second, the current calculated probabilities of error for all the bits in all the subcodes except the first (those labeled b_1 through b_{10} in Fig. 9); third, the original computed syndrome s_i, for all the subcodes except the first (s_2 and s_3 in Fig. 9). Then using Bayes law,

$$\frac{p_{i1}}{1-p_{i1}} = \frac{\Pr(\text{bit } i \text{ in error} | S_2, S_3) \Pr(S_2, S_3)}{\Pr(\text{bit } i \text{ not in error} | S_2, S_3) \Pr(S_2, S_3)}$$

$$= \frac{\Pr(S_2, S_3 | \text{bit } i \text{ in error}) p_i}{\Pr(S_2, S_3 | \text{bit } i \text{ not in error})(1-p_i)}.$$

Assuming all other bits in subcode two and subcode three are independent,

$$= \frac{\Pr(S_2 | \text{bit } i \text{ in error})}{\Pr(S_2 | \text{bit } i \text{ not in error})} \cdot \frac{\Pr(S_3 | \text{bit } i \text{ in error})}{\Pr(S_3 | \text{bit } i \text{ not in error})} \cdot \frac{p_i}{1-p_i}$$

$$= r_{i2} r_{i3} r_i.$$

Each of the ratios r_{ij} can be computed by computing the *a posteriori* probability of error for the ith bit given the constraints of the jth code and using Bayes law. Several methods for obtaining the required *a posteriori* probabilities are presented in [14]. Solving for p_{i1},

$$p_{i1} = \frac{r_{i2} r_{i3} r_i}{1 + r_{i2} r_{i3} r_i}.$$

By computing $r = r_{i1} r_{i2} r_{i3} r_i$ and then dividing out the jth term, p_{ij} can be computed for each subcode.

This procedure propagates information across the graph, bringing independent information towards each digit along all the paths of the tree emanating from that digit. (After $\lfloor (g-2)/4 \rfloor$ iterations the independence assumption breaks down.) At the end of the process, the final estimate for each bit is based on the stored original value and the computed probabilities of error of that value.

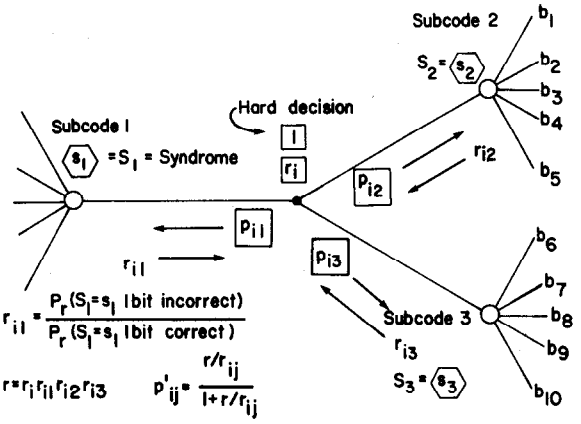

Fig. 9. Iteration step.

Since this probabilistic decoder incorporates error probabilities supplied by the receiver into the decoding process, it is able to take advantage of accurate channel models. The price paid for this generality is the need for fairly involved real arithmetic operations in both the subcode and digit phases.

The final algorithm we present will correct any pattern of $\lfloor (d_T - 1)/2 \rfloor$ or fewer errors, where d_T is the tree lower bound on minimum distance. Again for clarity we will treat the binary case. Referring to Fig. 7, it can be seen that the tree bound was derived by showing that any pattern of bits which satisfies all the subcodes in the tree, meaning *only* those above or at the level where the first minimum cycle is formed, must have minimum distance at least d_T. The actual code is a subset of the set of vectors of bits satisfying the tree subcodes, and therefore must have a minimum distance at least as large as d_T. The algorithm is designed to decode the bits at the top of the tree to those values occurring in the vector of bits that satisfies all the subcodes in the tree and requires the fewest changes in the actual received bits. The vector itself may not be a codeword.

Presentation of the algorithm will be facilitated by establishing a formal indexing for the registers required. Let R_{ij} be the register associated with bit i, $i = 1,2,\cdots,N$, that is accessed by subcode processor j, $j = 1,2,\cdots,S$, $R_{ij}(t)$ the value stored by the register after the tth iteration, and R'_{ij} a corresponding temporary storage register. Similarly let V_i, $i = 1,2,\cdots,N$, be a register storing a value $V_i(0)$ that is $+1$ or -1 if the ith bit was received as a 1 or 0 respectively. Finally let J_i be the index set of the subcode processors accessing bit i, and let I_j be the index set of bits accessed by subcode processor j.

Algorithm B:

Initialization: Each of the V_i, $i = 1,2,\cdots,N$, is loaded with a value of $+1$ or -1 supplied from the channel according to the received value 1 or 0, respectively, of the ith bit. Then for $i = 1,2,\cdots,N$, each register R_{ij}, $j \in J_i$ is assigned value $R_{ij}(0) = V_i(0)$.

Iterative loop: For $t = 1,2,\cdots,\lfloor (g-2)/4 \rfloor$ the following two phases are performed.

1) Subcode phase: For $j = 1,2,\cdots,S$ the jth subcode processor computes

$$R'_{ij} = \frac{1}{2}\left[\max_{\substack{C\in\mathcal{C} \\ c_i=+1}} \left(\boldsymbol{C}\cdot\boldsymbol{R}_j(t-1) - R_{ij}(t-1)\right) - \max_{\substack{C\in\mathcal{C} \\ c_i=-1}} \left(\boldsymbol{C}\cdot\boldsymbol{R}_j(t-1) + R_{ij}(t-1)\right)\right]$$

for each $i \in I_j$. Here $\mathcal{C}$ is the set of n-dimensional real vectors of plus and minus ones $\boldsymbol{C} = (C_{i1}, C_{i2},\cdots,C_{in})$ each derived from a codeword in the subcode by replacing each 1 in the codeword by $+1$ and each 0 by -1; $\boldsymbol{R}_j(t-1)$ is the ordered vector of register values $[R_{i_1 j}(t-1), R_{i_2 j}(t-1),\cdots,R_{i_n j}(t-1)]$ with $i_l \in I_j$; and $\boldsymbol{C}\cdot\boldsymbol{R}_j(t-1)$ denotes real vector inner product. If $g/2$ is odd, and $t = 1$, all R_{ij} values are divided by $1/m$ (to avoid multiply counting bits at the bottom of the tree).

2) Bit register phase: For each $i = 1,2,\cdots,N$, the registers for the ith bit are updated according to

$$R_{ij}(t) = \sum_{l\in J_i} R'_{il} - R'_{ij} + V_i(0).$$

Final Decision:

The jth subcode processor finds the vector for $\boldsymbol{C}$ that achieves $\max_{C\in\mathcal{C}}[\boldsymbol{C}\cdot\boldsymbol{R}_j(\lfloor (g-2)/4 \rfloor)]$ and sets $R'_{i_l j} = C_{i_l}$, the corresponding component of the maximizing subcode codeword. Then the output value of the ith bit is one if $\Sigma_{j\in J_i} R'_{ij} > 0$, zero otherwise. In other words, the bit's value is determined by a majority vote of the best final estimates provided by the subcodes checking it.

We will now prove that any pattern $\lfloor (d_T - 1)/2 \rfloor$ or fewer errors will be corrected by Algorithm B because in fact *all* the values c_{i_l} returned for a bit will be $+1$ if the bit was a one and -1 if it was a zero. The first step is the following lemma. Referring to Fig. 7, let the jth subcode node σ_j be the root and the ith bit β_i be an arbitrary bit at depth one. Let $T_{ij}(X)$ be the set of nodes η of the graph such that there is a minimum length path from σ_j to η passing through β_i of length l, $1 \le l \le X$.

Lemma 1: After t iterations of Algorithm B, $R_{ij}(t) = Z_{ij}(t) - U_{ij}(t)$ where:

$Z_{ij}(t)$ = minimum number of received bits $\beta \in T_{ij}(2t+1)$ that must be changed in order to have $\beta_i = 0$ and have all subcode nodes $\sigma \in T_{ij}(2t+1)$ be satisfied (i.e., the bits incident on σ form a codeword in σ).

$U_{ij}(t)$ = minimum number of received bits $\beta \in T_{ij}(2t+1)$ that must be changed in order to have $\beta_i = 1$ and have all subcode nodes $\sigma \in T_{ij}(2t+1)$ be satisfied.

Proof: The proof is by induction on the number of iterations. Clearly for $t = 0$ the lemma holds because $R_{ij}(0) = Z_{ij}(0) - U_{ij}(0) = V_i(0)$. Assuming it holds for $t-1$, a register $R_{i'j'}$ between depths two and three coming into subcode nodes $\sigma_{j'}$ for some $j' \in J_i$, $j' \neq j$, contains $Z_{i'j'}(t-1) - U_{i'j'}(t-1)$, $i' \in I_{j'}$. This can be recognized as the difference in changes needed in the subtree below

and including $\beta_{i'}$ to depth $2t+1$ for $\beta_{i'}$ to be zero or one, respectively. Then for the constraints of $\sigma_{j'}$ to be satisfied requires

$$Z = \min_{\substack{C\in\mathcal{C} \\ c_i=-1}} \left(\sum_{\substack{i'\neq i \\ c_{i'}=-1}} Z_{i'j'}(t-1) + \sum_{\substack{i'\neq i \\ c_{i'}=+1}} U_{i'j'}(t-1) \right)$$

changes below $\sigma_{j'}$ for $\beta_{i'}$ to be zero, and

$$U = \min_{\substack{C\in\mathcal{C} \\ c_i=+1}} \left(\sum_{\substack{i'\neq i \\ c_{i'}=-1}} Z_{i'j'}(t-1) + \sum_{\substack{i'\neq i \\ c_{i'}=+1}} U_{i'j'}(t-1) \right)$$

for $\beta_{i'}$ to be one. Let ${}^{-}\boldsymbol{C}$ be the vector achieving the former minimum and ${}^{+}\boldsymbol{C}$ the vector achieving the latter. The difference is

$$\begin{aligned} Z-U &= \sum_{\substack{i'\neq i \\ {}^{-}c_{i'}=-1,\,{}^{+}c_{i'}=+1}} \left(Z_{i'j'}(t-1)-U_{i'j'}(t-1)\right) \\ &\quad - \sum_{\substack{i'\neq i \\ {}^{-}c_{i'}=+1,\,{}^{+}c_{i'}=-1}} \left(Z_{i'j'}(t-1)-U_{i'j'}(t-1)\right) \\ &= (1/2)[{}^{+}\boldsymbol{C} - {}^{-}\boldsymbol{C}]\cdot\boldsymbol{R}_{j'}(t-1) - R_{ij'}(t-1). \end{aligned}$$

Now for any $\boldsymbol{C}$ with $c_i=-1$

$$\begin{aligned} &\boldsymbol{C}\cdot\boldsymbol{R}_{j'}(t-1) + R_{ij'}(t-1) \\ &\quad = \sum_{\substack{i'\neq i \\ c_{i'}=+1}} \left(Z_{i'j'}(t-1)-U_{i'j'}(t-1)\right) \\ &\qquad - \sum_{\substack{i'\neq i \\ c_{i'}=-1}} \left(Z_{i'j'}(t-1)-U_{i'j'}(t-1)\right) \\ &\quad = \sum_{i'\neq i} Z_{i'j'}(t-1) + \sum_{i'\neq i} U_{i'j'}(t-1) - 2 \\ &\qquad \cdot\left[\sum_{\substack{i'\neq i \\ c_{i'}=-1}} Z_{i'j'}(t-1) + \sum_{\substack{i'\neq i \\ c_{i'}=+1}} U_{i'j'}(t-1) \right]. \end{aligned}$$

Therefore, since the first term is independent of $\boldsymbol{C}$,

$$\max_{\substack{C\in\mathcal{C} \\ c_i=-1}} \left(\boldsymbol{C}\cdot\boldsymbol{R}_{j'}(t-1) + R_{ij'}(t-1)\right) = {}^{-}\boldsymbol{C}\cdot\boldsymbol{R}_{j'}(t-1) + R_{ij'}(t-1).$$

Similarly,

$$\max_{\substack{C\in\mathcal{C} \\ c_i=+1}} \left(\boldsymbol{C}\cdot\boldsymbol{R}_{j'}(t-1) - R_{ij'}(t-1)\right) = {}^{+}\boldsymbol{C}\cdot\boldsymbol{R}_{j'}(t-1) + R_{ij'}(t-1).$$

Thus

$$R'_{ij'} = (1/2)\left[\left({}^{+}\boldsymbol{C}\cdot\boldsymbol{R}_{j'}(t-1) + R_{ij'}(t-1)\right) - \left({}^{-}\boldsymbol{C}\cdot\boldsymbol{R}_{j'}(t-1) + R_{ij'}(t-1)\right)\right] = Z-U.$$

The desired difference of changes in subtree $T_{ij}(2t+1)$ is then

$$Z_{ij}(t) - U_{ij}(t) = \sum_{\substack{j'\in J_i \\ j'\neq j}} R'_{ij'} + V_i(0) = R_{ij}(t)$$

as required.

Theorem 6: Algorithm B will correct any pattern of $\lfloor (d_T-1)/2 \rfloor$ or fewer errors.

Proof: Again referring to Fig. 7, by Lemma 1 after $t_o = \lfloor (g-2)/4 \rfloor$ iterations the contents of the registers coming into the subcode at the top of the tree will be $\boldsymbol{R}_j(t_o) = \boldsymbol{Z}_j(t_o) - \boldsymbol{U}_j(t_o)$. Let $\boldsymbol{C}$ be the vector corresponding to the codeword in the subcode that was actually sent and let $\boldsymbol{C}'$ be the vector for any other codeword in the subcode. By the proof of the tree bound, we know that if no errors occur during transmission

$$\begin{aligned} (1/2)(\boldsymbol{C}-\boldsymbol{C}')\cdot\boldsymbol{R}_j(t_o) &= \sum_{\substack{c_i=+1 \\ c'_i=-1}} \left(Z_{ij}(t_o)-U_{ij}(t_o)\right) \\ &\quad - \sum_{\substack{c_i=-1 \\ c'_i=+1}} \left(Z_{ij}(t_o)-U_{ij}(t_o)\right) \geq d_T. \end{aligned}$$

It is obvious that changing the received value for a single bit in the subtree $T_{ij}(2t_o+1)$ can at most change $Z_{ij}(t_o) - U_{ij}(t_o)$ by two. Therefore, if fewer than $\lfloor (d_T-1)/2 \rfloor$ errors have occurred

$$(1/2)(\boldsymbol{C}-\boldsymbol{C}')\cdot\boldsymbol{R}_j(t_o) \geq d_T - 2\lfloor (d_T-1)/2 \rfloor \geq 1.$$

Thus $\boldsymbol{C}$ achieves $\max_{C\in\mathcal{C}}[\boldsymbol{C}\cdot\boldsymbol{R}]$ and the value c_i assigned to each bit will be $+1$ if the transmitted bit was one, -1 if the transmitted bit was zero. Since this holds no matter which subcode is at the top of the tree, the final majority vote must give the correct value for each bit.

This same algorithm can easily be modified to take advantage of channel reliability information coming from a binary channel. Rather than loading the registers for a bit with $+1$ or -1 from a hard decision receiver, the registers can be loaded with the logarithm of the likelihood ratio for one versus zero. Algorithm B will then decode for a maximum likelihood version of the tree bound. Also, a similar algorithm can be devised for a nonbinary channel, but it will require more registers for each digit.

An immediate criticism that this algorithm invites is that it requires a maximum likelihood decoder for the subcodes. While this admittedly has dire implications for the complexity resulting from the use of an arbitrary subcode, the requirement is not as bad as it may seem for many recursively defined codes. First, a maximum likelihood decoder for a simple parity check is not complex, and this algorithm can thus easily be used on any low-density parity-check code. Second, for fixed subcodes the complexity of the algorithm grows only linearly in the girth of the graph; thus, for codes based on a very large girth graph, the complexity of the subcode decoder is not a major issue. Finally, it may be possible to get good decoding performance from a similar algorithm which works recursively, namely the subcode decoding is performed by an algorithm of the same type which computes only an approximation to the true maximum likelihood value of $\boldsymbol{Z}_j(t_o) - \boldsymbol{U}_j(t_o)$.

VII. Decoding Complexity

The decomposition of a decoding process into a series of subcode operations requiring only local information

guarantees that the asymptotic growth of complexity for a sequence of codes of fixed rate is very low, as we will now argue by deriving upper bounds on the complexity for two specific sequences.

As a first case for analysis, consider the effect of using a single recursive step, basing a sequence of codes on a single subcode and a sequence of graphs of fixed subcode node degree and fixed digit degree but increasing girth. Although the exact rate of the codes depends on the technique for assigning digit positions in each of the subcodes, Theorem 1 ensures that the rate of any of the codes has a fixed lower bound. For any of these codes, the number of subcodes in the graph is proportional to the code length, as mentioned in Theorem 1. It follows immediately that the computational complexity of error *detection* is $O(N)$, because detection requires only a detection check by each of the subcodes involving a fixed number of operations. Since the detection procedure admits total parallelism, the time complexity is obviously $O(1)$.

Now consider the error-correcting Algorithm B used for Theorem 6. Each iteration requires a number of arithmetic operations proportional to N: the procedure carried out at each subcode node involves a fixed number of arithmetic operations per node, as does the set of operations carried out at each digit node. Using Gallager's graph construction, the girth of the graph is proportional to $\log N$. If, as is the case for Algorithm B, the number of iterations is held proportional to the girth, it follows that the computational complexity in terms of arithmetic operations is $O(N \log N)$. However, the registers required for the algorithm must store a number of the size of the minimum distance of the code, which for the best possible codes of this form, may grow linearly in N. This could appear to imply that the complexity of each register operation itself grows with N. On the other hand, the values stored in the registers are used only in comparisons with other registers. Consequently, by appropriately scaling the values in all of the registers at each iteration, the comparative accuracy can be maintained up to any desired level with registers of fixed size (see Appendix B). The $O(N \log N)$ bound is then preserved. Again, the complete parallelism gives the algorithm a time complexity of $O(\log N)$.

A second avenue to the construction of a sequence of codes is to use graphs of the same girth and digit node degree at each stage of recursion and create each new code in the sequence by using an additional recursion. Assuming the lower bound on rate holds at each stage, however, the rate of the base subcode must be increased with the number of stages of recursion to keep the rate of the final code fixed. As a specific illustration, consider forming a sequence using the product code graph which has digit degree two, girth eight, and causes the codelength to be squared. Let $T(N, P)$ be the number of operations required to decode a code of length N and redundancy rate $P = (N - K)/N$ using a recursive algorithm similar to Algorithm B. A two-phase process is iterated $g/4$ times, but here the subcode node decoders themselves make iterated recursive calls to a similar algorithm for the subcodes, on down to the base code. If there are s stages of recursion, $N = n^{2^s}$, where n is the length of the base code. The complexity satisfies the recursion relation

$$T(N, P) = g/4\left(mN/\sqrt{N}\right)T\left(\sqrt{N}, \frac{P}{m}\right) + cN$$

$$= 4\sqrt{N}\,T\left(\sqrt{N}, \frac{P}{2}\right) + cN,$$

where cN is the number of operations required for the bit register interchanges and final votes. Solving for $T(N, P)$ in terms of the base code complexity $T(n, p)$ gives

$$T(N, P) = 4^s N^{1-(1/2^s)} T\left(N^{1/2^s}, \frac{P}{2^s}\right) + \left(\sum_{i=0}^{s-1} 4^i\right) cN$$

$$\leq \left(4^{\log(\log N/\log n)} N/n\right) T\left(n, \frac{P \log n}{\log N}\right)$$

$$+ \left(4^{\log(\log N/\log n)}/3\right) cN$$

$$\leq \frac{(\log N)^2}{(\log n)^2} N\left[\frac{1}{n} T\left(n, \frac{P \log n}{\log N}\right) + \frac{c}{3}\right].$$

If the lower bound on rate is satisfied at every stage of recursion, the base subcode used for the successive constructions must have a rate increasing toward one. For instance, when the base subcodes are binary Hamming codes decoded with a decoding algorithm that requires $O(n^2)$ operations, the ultimate complexity can be derived as follows: for a Hamming code $p = \log n/n$, implying $n = \log N/P$ and $\log n = \log\log N - \log P$. Thus

$$T\left(n, \frac{P \log n}{\log N}\right) = T\left(n, \frac{\log n}{n}\right) = O(n^2) = O\left(\frac{(\log N)^2}{P^2}\right).$$

Therefore for fixed P,

$$T(N, P) \leq \frac{(\log N)^2}{(\log\log N - \log P)^2} \cdot N\left[\frac{P}{\log N} O\left((\log N)^2\right) + \frac{c}{3}\right]$$

and $T(N, P) = O(N(\log N)^3/(\log\log N)^2)$. Again the parallelism ensures that the time complexity of this process is no worse than $O((\log N)^3/(\log\log N)^2)$.

By using graphs of larger girth in the same type of recursion, the exponent of the $\log N$ factor can be reduced. Furthermore, all of the operations are of the same form, namely repeated operations carried out by the base code processors and digit processors. The complexity of the machine architecture of the decoder lies not in the actual computations but in the data movements or wiring that allow the processor to access the appropriate subsets of the data.

VIII. An Example

Many of the ideas of the previous sections can be illustrated with a two-stage recursive construction of a [55, 11, 18] code starting from a simple parity-check base code.

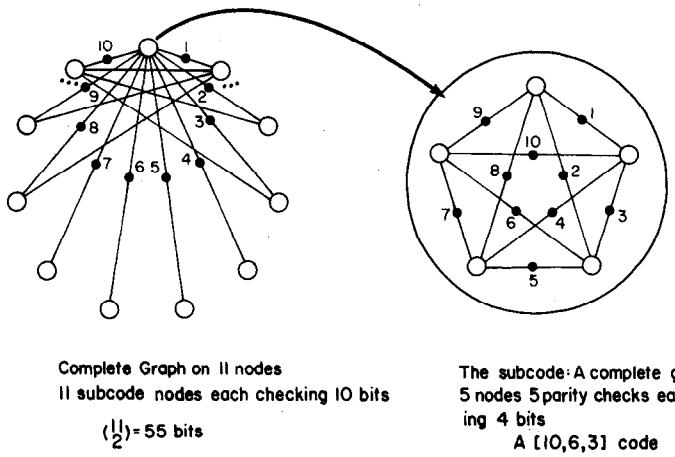

Fig. 10. A [55, 11, 18] recursive code.

The graph used for each stage is a bipartite graph constructed from a simple complete graph.

Starting from a complete graph on five nodes, create a bipartite graph by inserting a bit node in the middle of each edge, as shown in Fig. 10. Each of the original nodes can be viewed as a subcode node checking four bits. Let each of these subcodes correspond to a simple parity check. The result is the standard [10, 6, 3] graph code. The tree bound for this simple construction guarantees minimum distance three, the true distance. The lower bound on the rate of the code is 1/2, but one of the parities is dependent. Now use this code as the subcode in similar construction starting from a complete graph on 11 nodes. The new graph has $\binom{11}{2} = 55$ bit nodes, and each subcode checks ten bits. Because the subcode has minimum distance three, the tree bound guarantees a minimum distance of at least six for the new code, no matter how the positions of the ten bits checked are assigned in the subcode. The particular assignment used for the top subcode is shown in Fig. 10; the ten bits checked by that subcode, labelled one through ten in the eleven subcode node graph, are given the positions labelled one through ten in the graph for the subcode. The assignments for the other subcodes are then determined by cyclic symmetry in the large graph: the assignments for any subcode can be determined by rotating it to the top subcode position. The minimum distance of the new code, determined by computer, is 18, far surpassing the tree bound. A minimum weight word is shown in Fig. 11.

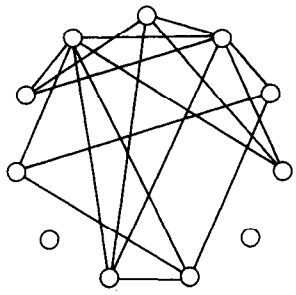

Fig. 11. A weight 18 word.

A variety of encoders for the code can be constructed; we will briefly sketch the principles underlying three different versions. The space of codewords is spanned by the ten cyclic shifts of the word shown in Fig. 11 plus the all ones word. Another basis for the space is the set of eleven cyclic shifts of a word that has only one of the outermost bits nonzero (e.g., the bits labelled one and ten in Fig. 10). The outermost bits thus constitute an information set for the code. The first encoder, somewhat analogous to an encoder for a normal cyclic code based on the generator polynomial, simply adds a one into every parity bit which is one in the basis vector when the corresponding information bit is one. For this particular code this would be a cumbersome architecture requiring four eleven bit shift registers to store the parity bits and a large number of adders. For similar codes of higher rate, it could be more advantageous.

A second encoder can be derived from the idea of propagation of a solution via subcode encodings discussed in Section IV. Since this code is a two-stage recursive construction, the concept of a propagating solution can be carried down to the original parity checks. Referring to Fig. 10, suppose the outermost eleven bits are given. Suppose the values for the bits on the second tier (e.g., bits two and nine) are computed using some general technique. Now consider the subcode diagram. Since bits one, ten, two, and nine are known, the top parity-check alone determines bit eight. Similarly, in another subcode a single parity check determines bit three. But this implies that bit four can be determined, and another subcode gives bit seven, which in turn allows six and then five to be computed. A parallel implementation of this strategy can compute the codeword in five steps.

The third encoder we suggest is analogous to the parity-check polynomial encoder for a standard cyclic code. Suppose the information bits have been loaded into an 11 bit cyclic shift register. Any bit on the second tier can be computed by summing a subset of the information bits in the cyclic register; since the graph has cyclic symmetry, all the second tier bits can be computed by cyclically shifting the register. The bits of the remaining three tiers can be obtained in a similar manner by using three different

summing circuits connected to the appropriate stages of the information bit shift register. Clearly the codeword can be computed five bits at a time with an encoder requiring only 11 bits of memory and four different summing circuits.

A good transmission order for enhancing the code's effectiveness against burst errors is shown in Fig. 12. It is easy to see why bursts will tend to be corrected: for any burst of length 11 or less most of the errors appear as single errors in separate subcodes.

A modified version of Algorithm B was implemented and tested on a general purpose computer. Rather than decoding at the subcode level, the algorithm works only at the sub-subcode level, that is, on the parity checks themselves. Each of the eleven subcodes is built on a graph using five parity checks. Interpreted at this level, there are 55 parity checks each checking four bits, and 55 bits each checked by four parities. The two-phase algorithm used four registers for each bit; all four were loaded with a $+1$ if the bit was a one, -1 if the bit was a zero. In the subcode phase the four registers of the bits checked by a parity were updated according to the generic prescription

$$R_1' = R_1(t-1) + \operatorname{sgn}\left(R_2(t-1)\right) \cdot \operatorname{sgn}\left(R_3(t-1)\right) \operatorname{sgn}\left(R_4(t-1)\right) \cdot \min\left(|R_2(t-1)|, |R_3(t-1)|, |R_4(t-1)|\right).$$

In the bit phase the four registers for the bit were updated according to

$$R_1(t) = \left\lceil \frac{R_2' + R_3' + R_4'}{2} \right\rceil.$$

After a specified number of iterations the output value of a bit was determined by the sign of the sum of the associated registers, one if the sum was postive, zero if the sum was negative.

By testing the algorithm against five consecutive shifts of each burst error pattern containing i errors, $i = 1, 2, \cdots, 11$, we determined that the algorithm will correct all bursts of length 11 or less. In preliminary empirical testing against random errors, the same algorithm corrected all randomly generated patterns of seven or fewer errors and many of greater weight. All weight six error patterns tested were decoded in five or fewer iterations; one weight seven required 22 iterations.

IX. Conclusion

The recursive approach to the construction of error-correcting codes which we have presented offers hope that very efficient codes can be implemented with relatively simple circuits and decoded by algorithms requiring only $O(N \log N)$ operations of fixed type. For this hope to become provable reality will require that the results of the current paper be strengthened by improved versions of the tree bound and a corresponding proof that low-complexity algorithms can decode up to the actual minimum distance, Theorems 2 and 6, respectively. All the evidence provided by our study of specific examples indicates that best codes

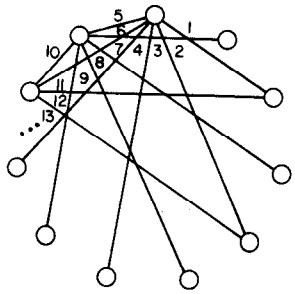

Fig. 12. Transmission order for burst-error correction.

of this form do indeed have minimum distances far greater than that guaranteed by the tree bound, and that the simple algorithms are effective.

Is there reason to believe that codes with this structure cannot be good? For the codes formed from a fixed subcode using large graphs, a partial answer to this question is provided by the upper bound on rate for low-density codes [8, pp. 39–40] which shows that the rate of the code must be bounded away from channel capacity for the probability of error to approach zero. From a practical viewpoint, however, this limitation is insignificant. A closely related argument is that the strength of such codes against burst errors, which is a direct consequence of the decomposition into subcodes, must preclude optimal effectiveness against random errors. On the other hand, as the previous example illustrates, the best codes appear to have the potential for correcting almost as many random errors as burst errors.

Using Gallager's low-density codes, Zyablov and Pinsker [15] have shown that there exist codes capable of correcting all error patterns of weight less than αN, for a fixed positive constant α, in $O(N \log N)$ operations. The proof necessitates a fairly dramatic loss of rate, and it may well be that some loss of rate is necessary to maintain low complexity even when using our approach.

On the other hand, the potential advantages of these codes may well compensate some slight loss of rate. The low complexity may allow the use of much longer block lengths than can be realistically contemplated with most existing techniques, and more accurate representations of the information provided by the receiver, such as bit reliability information, can be effectively incorporated into the decoding process. In addition, the natural parallelism in the computation is well suited to implementations using a large number of large scale integrated circuits of a single design.

Although we have presented the central ideas in the context of block codes, the same concepts are applicable to convolutional codes as well, as we hope to discuss in a later paper.

Acknowledgment

The author wishes to thank Peter Wilde, formerly a senior at UCSC, for carrying out most of the computer studies on the examples presented and for suggesting the transmission order for the [55, 11, 18] code.

APPENDIX A

Probability Bound for the Subcode Correction Procedure of Algorithm A

Suppose that a vector of errors is added to a codeword from a Hamming code of length $2^m - 1$ such that one bit in the word has probability q of being in error and the remaining bits each have probability p of being in error, all independently. By the symmetry of the code, we can assume without loss of generality that the first bit in the code is in error with probability q. Let the word be corrected according to the standard single error correction procedure, and let p' be the probability that the first bit is in error after the correction.

Lemma 2: If $m > 2$ and $q = 1/(2^m - 2)$ there exists an $r < 1$ such that $p' \le rp$ for $0 \le p \le q$.

Proof:

$$\begin{aligned} p' &= q\,\text{Prob (first bit not changed/first bit incorrect)} \\ &\quad + (1-q)\,\text{Prob (first bit changed/first bit correct)} \\ &\triangleq qQ_1 + (1-q)Q_o \\ Q_1 &= 1 - \text{Prob (first bit changed/first bit incorrect)} \\ &= 1 - \text{Prob (errors in remaining bits form a codeword)} \\ &\triangleq 1 - P_1. \end{aligned}$$

That is, the incorrect first bit will be perceived as a single error if and only if the errors in the rest of the bits form a codeword. Note also that $Q_o = \text{Prob}$ (errors in remaining bits plus an error in the first bit form a codeword).

To obtain an expression for the latter, consider a length $2^m - 1$ word in which *all* bits have probability of error p. Let P_o be the probability the errors form a codeword. Then $P_o = pQ_o + (1-p)P_1$, and so $Q_o = (P_o - (1-p)P_1)/p$. Substituting,

$$\begin{aligned} p' &= q(1-P_1) + (1-q)(P_o - (1-p)P_1)/p \\ &= q + (1/p)(1-q)(P_o - P_1) + (1-2q)P_1. \end{aligned}$$

Let $A(z)$ by the weight enumerator for the Hamming code

$$A(z) = \sum_{i=0}^{2^m-1} A_i Z^i,$$

where A_i is the number of codewords of weight i. Then

$$P_o = (1-p)^{2^m-1} A\left(\frac{p}{1-p}\right).$$

Using the MacWilliams identity, $A(z)$ can be obtained from the weight enumerator for the dual of the Hamming code, $B(z)$ [16, pp. 400–407]:

$$A(z) = 2^{2^m-1-m}\left(\frac{1+z}{2}\right)^{2^m-1} B\left(\frac{1-z}{1+z}\right),$$

$$B(z) = 1 + (2^m - 1)z^{2^{m-1}}.$$

Thus

$$P_o = 2^{-m}\left(1 + (2^m - 1)(1-2p)^{2^{m-1}}\right).$$

Similarly P_1 is the probability errors occurring with probability p form a codeword in the shortened code whose null matrix is the Hamming code null matrix without the first column. The dual of this code has weight enumerator

$$1 + 2^{m-1}z^{2^{m-1}} + (2^{m-1} - 1)z^{2^{m-1}}.$$

Therefore

$$P_1 = 2^{-m}\left(1 + 2^{m-1}(1-2p)^{2^{m-1}-1} + (2^{m-1} - 1)(1-2p)^{2^{m-1}}\right).$$

Then

$$\begin{aligned} P_o - P_1 &= 2^{-m}\left(1 + (2^{m-1})(1-2p)^{2^{m-1}} - 1 \right. \\ &\qquad \left. -2^{m-1}(1-2p)^{2^{m-1}-1} - (2^{m-1} - 1)(1-2p)^{2^{m-1}}\right) \\ &= 2^{-m}\left(2^{m-1}(1-2p)^{2^{m-1}} - 2^{m-1}(1-2p)^{2^{m-1}}\right) \\ &= -p(1-2p)^{2^{m-1}-1}. \end{aligned}$$

Substituting,

$$\begin{aligned} p' &= q - (1-q)(1-2p)^{2^{m-1}} \\ &\quad + (1-2q)2^{-m}\left(1 + 2^{m-1}(1-2p)^{2^{m-1}-1} \right. \\ &\quad \left. + (2^{m-1} - 1)(1-2p)^{2^{m-1}}\right). \end{aligned}$$

Now we will show that $p' - p \le 0$ if $q = 1/(2^m - 2)$. Rearranging terms,

$$\begin{aligned} p' - p &= q + 2^{-m}(1-2q) - \frac{(1-2p)^{2^{m-1}-1}}{2} \\ &\quad + (1-2p)^{2^{m-1}}(1-2q)2^{-m}(2^{m-1} - 1) - p. \end{aligned}$$

It is easy to check that $p' - p = 0$ at $p = 0$. Taking the derivative

$$\begin{aligned} \frac{d}{dp}(p' - p) &= (2^{m-1} - 1)(1-2p)^{2^{m-1}-2} - 2^{m-1} \\ &\quad \cdot (1-2p)^{2^{m-1}}(2)(1-2q)2^{-m}(2^{m-1} - 1) - 1 \\ &= (2^m - 2)(1-2p)^{2^{m-1}-2}(q + p(1-2q)) - 1. \end{aligned}$$

Setting $q = 1/(2^m - 2)$, at $p = 0$,

$$\left.\frac{d}{dp}(p' - p)\right|_{p=0} = (2^m - 2)\frac{1}{2^m - 2} - 1 = 0.$$

Furthermore, with $q = 1/(2^m - 2)$ the second derivative is negative for all $0 < p \le 1/(2^m - 2)$:

$$\begin{aligned} &\frac{d^2}{dp^2}(p' - p) \\ &= (2^m - 2)\left[(2^{m-1} - 2)(1-2p)^{2^{m-1}-3} \right. \\ &\quad \left. \cdot(-2)(q + p(1-2q)) + (1-2p)^{2^{m-1}-2}(1-2q)\right] \\ &= (2^m - 2)(1-2p)^{2^{m-1}-3} \\ &\quad \cdot\left[(1-2p)(1-2q) - (2^m - 4)(q + p(1-2q))\right] \\ &= (2^m - 2)(1-2p)^{2^{m-1}-3} \\ &\quad \cdot\left[-p(1-2q)(2^m - 2) + (1-2q) - (2^m - 4)q\right] \\ &= (2^m - 2)(1-2p)^{2^{m-1}-3}\left[-p(2^m - 4)\right] \le 0 \end{aligned}$$

for $m > 2$. By using Taylor's theorem with the remainder, it follows that $p'/p \le 1$ for $0 \le p \le q$ when $q = 1/(2^m - 2)$. To complete the proof, consider

$$\frac{\partial}{\partial q}\left(\frac{p'}{p}\right) = \frac{(2^m - 2)}{2^m}\,\frac{(1 - (1-2p)^{2^{m-1}}}{p}.$$

Since this is positive and independent of q for all $p > 0$, by decreasing q from $1/(2^m - 2)$ we will decrease p'/p. Moreover,

$$\lim_{p \to 0} \frac{\partial}{\partial q}\left(\frac{p'}{p}\right) = \frac{2^m - 2}{2^m} 2^m = 2^m - 2.$$

Therefore the minimum of $(\partial/\partial q)(p'/p)$ over $0 \le p \le 1/(2^m - 2)$ is greater than zero and $q < 1/(2^m - 2)$ implies that there exist an $r < 1$ such that $(p'/p) < r$.

Appendix B

Algorithm B with Registers of Fixed Size

Theorem A2: Algorithm B will correct any pattern of $\left\lfloor \left(d_T(1 - x^{-(l-2)}) - 1\right)/2 \right\rfloor$ using registers storing signed, l digit, integers represented base x, $x = (d - 1)(m - 1)$ for $x > 1$.

Proof: During the tth iteration, the digits stored in a register are the l coefficients a_i, of the base x expansion of the rounded-off value

$$R_{ij}(t) = a_o x^t + a_1 x^{t-1} + \cdots + a_{l-1} x^{t-l+1}$$
$$= Z_{ij}(t) - U_{ij}(t) + \epsilon_{ij}(t).$$

It is easy to prove by induction that the error, $\epsilon_{ij}(t)$, must be bounded by $|\epsilon_{ij}(t)| \le x^{t-l+1}$ as follows: clearly when the register is initialized rounding off the value guarantees that $|\epsilon_{ij}(0)| < x^{-(l-1)}$. By the assumptions of the tree bound, the difference can increase at most by a factor of $(d - 1)(m - 1)$ as it moves up one level in the tree. Since $x = (d - 1)(m - 1)$ the registers are guaranteed not to overflow unless minimum distance exceeds the tree bound. Similarly, $\max_{i,j} \epsilon_{ij}(t) \le (d - 1)(m - 1) \max_{ij} \epsilon_{ij}(t - 1)$ and $|\epsilon_{ij}(t - 1)| \le x^{t-l}$ for all i and j implies that $|\epsilon_{ij}(t)| \le x^{t-l+1}$ for all i and j. Consequently, if no errors occur during transmission,

$$(1/2)(\boldsymbol{C} - \boldsymbol{C}') \cdot \boldsymbol{R}_j(t_o) \ge d_T - d x^{t_o - l + 1} \ge d_T(1 - x^{-(l-2)}).$$

Therefore, if fewer than $\left\lfloor \left(d_T(1 - x^{-(l-2)}) - 1\right)/2 \right\rfloor$ errors have occurred

$$(1/2)(\boldsymbol{C} - \boldsymbol{C}') \cdot \boldsymbol{R}_j(t_o) \ge 1,$$

and as before the errors will be corrected.

References

[1] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. New York: North-Holland, 1977.

[2] P. Elias, "Error-free coding," *IRE Trans. Inform. Theory*, vol. PGIT-4, pp. 29–37, Sept. 1954.

[3] W. C. Gore, "Further results on product codes," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 446–451, July 1970.

[4] G. D. Forney, *Concatenated Codes*. Cambridge: MIT Press, Res. Monograph 37, 1966.

[5] S. M. Reddy, "On decoding iterated codes," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 624–627, Sept. 1970.

[6] E. M. Kasahara, Y. Sugiyama, S. Hirasawa, and T. Namekawa, "New classes of binary codes constructed on the basis of concatenated codes and product codes," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 462–468, July 1976.

[7] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[8] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA: MIT Press, 1963.

[9] W. W. Peterson and E. J. Weldon, Jr., *Error-Correcting Codes*. Cambridge, MA: MIT Press, 1972.

[10] P. J. Cameron and J. H. Van Lint, *Graph Theory, Coding Theory, and Block Designs*. Cambridge: Cambridge Univ. Press, 1975.

[11] P. Dembowski, *Finite Geometries*. New York: Springer, 1968.

[12] A. J. Hoffman and R. R. Singleton, "On Moore graphs with diameters 2 and 3," *IBM J. Res. Dev.*, vol. 4, pp. 497–504, Nov. 1960.

[13] Y. Wallach and V. Konrad, "On block-parallel methods for solving linear equations," *IEEE Trans. Comput.*, vol. C-29, pp. 354–359, May 1980.

[14] G. Battail, M. C. Decouvelaere, and P. Godlewski, "Replication decoding," *IEEE Trans. Inform Theory*, vol. IT-25, pp. 332–345, May 1979.

[15] V. V. Zyablov and M. S. Pinsker, "Estimation of the error-correction complexity of Gallager low-density codes," Probl. Peredach. Inform., vol. 11, pp. 23–36, Jan. 1975.

[16] E. Berlekamp, *Algebraic Coding Theory*. New York: McGraw–Hill, 1968.