Chapter 2
Theoretical Background

In this chapter, the theoretical background is presented covering design and construction of AG codes for the encoder and decoder along with important parameters. We also present a block diagram of the modified Sakata's algorithm for the first time. It shows how the construction of AG codes using Hermitian codes is performed using a hard-input hard-output (HIHO) decoding algorithm. Fundamentals of TCs encoder, decoder and interleaver design are shown. Examples of the construction of BTCs are also presented.

2.1 Algebraic Geometric Codes

For a long time researchers attempted to realize a very long non-binary block code with high code rate and large Hamming distance, however fulfilling these properties by classical codes was not possible. In 1981, V. D. Goppa showed a way to construct these codes which are now called Goppa codes or AG codes [1]. Goppa explained the construction from affine points of an irreducible projective curve and a set of rational functions defined on that curve. The famous Reed-Solomon (RS) code represents the best and for most the simplest example that demonstrates the construction of AG codes though it is constructed from the affine points of a projective line not a projective curve which is the case of Goppa codes.

The number of affine points determines the length of an AG code, so the cardinality of the chosen field restricts the length of RS codes which result in relatively short code lengths. Replacing the projective line with a projective curve yields more affine points which means longer code lengths while keeping the same size of the finite field [2, 3]. The longest possible codes can be obtained by choosing curves that have the maximum number of affine points which are called maximal curves, so the objective is always to find those curves whenever possible.

A possible reason that AG codes have not been studied and investigated very well is that they require a good knowledge of the theory of algebraic geometry, a difficult

J. A. Alzubi et al., *Forward Error Correction Based On Algebraic-Geometric Theory*,
SpringerBriefs in Electrical and Computer Engineering,
DOI: 10.1007/978-3-319-08293-6_2, © The Author(s) 2014

and complicated branch of mathematics. To overcome the previously stated problem, a simplified construction method was introduced in 1989 by Justesen et al. [4]. His method requires a basic understanding of algebraic geometry to produce AG codes. Although a limited number of AG codes—which is considered as a drawback—can be constructed using this method compared with using a more complicated AG approach, however this limited number of codes is still acceptable.

2.1.1 Construction of AG Code Parameters

According to Carrasco [5], an AG code can be constructed using Justesen's simplified method by choosing an irreducible affine smooth curve over a finite field. Classes of good curves that could be used to produce good AG codes are the Hermitian curves, elliptic curves, hyperelliptic curves, and so on, as they all have one point at infinity.

However, Hermitian curves with degree $m = r + 1$ where $r = \sqrt{q}$ are well known from the previous classes of curves and most popular for constructing AG codes defined over a finite field F_q[4]:

$$C(x, y) = x^{r+1} + y^r + y \tag{2.1}$$

To define the message length (k) and the designed minimum Hamming distance (d^*), all affine points (the points causing the curve to vanish) as well as the point at infinity on the chosen curve must be found. The number of the affine points which satisfy $C(x, y) = 0$ is $n = r^3$. Hasse-Weil bound gives an upper bound for the number of affine points n [4]:

$$n \leq 2\gamma\sqrt{q} + 1 + q \tag{2.2}$$

where γ is the genus of the curve.

It is worth giving a complete explanation of the curve genus as it is difficult to find a detailed simplified definition and method of genus computation. The genus is the maximum number of cuttings along non-intersecting simple curves [6]. The process of computing it is perhaps more interesting. Assume there exists a plane curve called C which is defined by $f(x, y) = 0$ where $f(x, y)$ is a two-dimensional polynomial composed of two variables. The degree of this polynomial is m which is the largest sum of the exponents of x and y in each term of the curve equation. Then the genus of C is:

$$\gamma = \frac{(m-1)(m-2)}{2} \tag{2.3}$$

if and only if C is non-singular curve.

A nonsingular curve, also called smooth curve, is the one which has no singular points. A singular point is defined as the point where something unusual happens on the curve like a sharp corner ($y^2 = x^3$) or a crossing of two branches ($y^2 = x^3 + x^2$).

Otherwise, when the curve has a finite number of singular points, it is called a singular curve [7].

As Hermitian curves saturate the Hasse-Weil bound, they called maximal curves making them suitable to generate long AG codes. Justesen's construction method suggests a non-negative integer j which is bounded by [8]:

$$m - 2 \leq j \leq \left| \frac{n-1}{m} \right| \tag{2.4}$$

Goppa or AG codes are of two types: functional Goppa codes (C_L) and residue Goppa codes (C_Ω). The latter is the dual of the former. In both types, the block length is equal to the number of affine points on the curve (n) [5]. To compute the length of the message for an AG code, a set of rational functions with a pole of order equal or less than the degree of the divisor (a) at the point at infinity (Q) must be found first [6], where the degree of the divisor is limited to be greater than $2\gamma - 2$ and less than n ($2\gamma < a < n$). In Justesen's simplified construction method $a = mj$. This set of rational functions is also called the linear space of aQ which is denoted by $L(aQ)$.

The number of elements in the previous set is equal to the message length k. It is called the dimension of aQ and denoted by $l(aQ)$ [8]. The Riemann-Roch theorem is used to calculate $l(aQ)$ [9, 10] which defines the message length k in functional Goppa codes $C_L(D, aQ)$ as:

$$k = l(aQ) = deg(aQ) - \gamma + 1 = a - \gamma + 1 \tag{2.5}$$

while the message length k in residue Goppa codes is defined by:

$$k = n - l(aQ) = n - a + \gamma - 1 \tag{2.6}$$

For both types of AG codes, a lower bound of the Hamming distance of AG codes is calculated and called the designed minimum Hamming distance d^* as the Hamming distance (d) cannot always be calculated accurately. Meeting the singleton bound when calculating minimum Hamming distance is required as the value will then be optimal [11]:

$$d = n - k + 1 \tag{2.7}$$

However, the main disadvantage that must be mentioned regarding the use of AG codes is that the designed minimum Hamming distance is affected inversely by the genus of the curve. This means that the larger the genus, the smaller the designed minimum Hamming distance, and vice versa. In contrast, the case of RS codes are constructed over an affine line of degree one and genus equal to zero [5].

So the actual designed minimum Hamming distance is [8, 10]:

$$d^* = n - k - \gamma + 1 \tag{2.8}$$

To compute the designed minimum Hamming distance for functional Goppa codes, we substitute the value of message length for those codes into (2.8) and get that:

$$d^* = n - a \tag{2.9}$$

Also for residue Goppa codes, the designed minimum Hamming distance can be found by substituting the message length k into (2.8) and get:

$$d^* = a - 2\gamma + 2 \tag{2.10}$$

As Justesen's simplified code have the same parameters as residue Goppa codes since $a = mj$, then the code parameters are [8]:

$$K = n - mj + \gamma - 1 \tag{2.11}$$

$$d^* = mj - 2\gamma + 2 \tag{2.12}$$

and the codeword length n is equal to the number of affine points on the curve as mentioned earlier.

2.1.2 Designing AG Encoder

To generate a generator matrix for an AG code, all the points that satisfy the chosen curve must be found which means all the points that make $C(x, y) = 0$ excluding the point at infinity. For Harmitian curves, as previously said, the number of these points is equal to $n = r^3$ where $r = \sqrt{q}$, and q is the finite field size [8].

A k two variables monomial basis is defined as: $F = x^a y^b$ where $0 \leq a < m$ and $b \geq 0$ and ordered using total graduated degree ($<_T$). This method of ordering follows the pattern: first-degree pair $(a, b) = (0, 0)$; next-degree pair (a', b') is [12]:

$$(a', b') = \begin{cases} (a - 1, b + 1) & \text{if } a > 0 \\ (b + 1, 0) & \text{if } a = 0 \end{cases} \tag{2.13}$$

So, degree pairs ordering is: $(0, 0) <_T (1, 0) <_T (0, 1) <_T (2, 0) <_T (1, 1) <_T (0, 2) <_T (3, 0) <_T (2, 1) <_T (1, 2) <_T (0, 3) <_T (4, 0) <_T (3, 1) <_T (2, 2) \ldots$

This gives monomial basis (ϕ_i):

$$\left\{1, x, y, x^2, xy, y^2, x^3, x^2y, xy^2, y^3, x^4, x^3y, x^2y^2, xy^3, y^4, x^5, \ldots\right\} \tag{2.14}$$

It is worth explaining another ordering technique which is called partial ordering as it will help to show the concrete difference between the two ordering techniques and will be helpful in understanding steps of the decoding procedure later on. Assume there are two pairs of integers $a = (a_1, a_2)$ and $b = (b_1, b_2)$ then [12]:

$$a < b \quad \text{if} \quad a_1 \leq b_1 \wedge a_2 \leq b_2 \wedge a \neq b \tag{2.15}$$

To obtain the final non-systematic generator matrix of the code, each of the monomial basis ϕ_i, $i = 1, 2, ..., k$ in $L(aQ)$ is evaluated at each affine point as the following:

$$G = \begin{bmatrix} \phi_1(p_1) & \phi_1(p_2) & \cdots & \phi_1(p_{n-1}) & \phi_1(p_n) \\ \phi_2(p_1) & \phi_2(p_2) & \cdots & \phi_2(p_{n-1}) & \phi_2(p_n) \\ \phi_3(p_1) & \phi_3(p_2) & \cdots & \phi_3(p_{n-1}) & \phi_3(p_n) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \phi_{k-1}(p_1) & \phi_{k-1}(p_2) & \cdots & \phi_{k-1}(p_{n-1}) & \phi_{k-1}(p_n) \\ \phi_k(p_1) & \phi_k(p_2) & \cdots & \phi_k(p_{n-1}) & \phi_k(p_n) \end{bmatrix} \tag{2.16}$$

Extracting the original message from the decoded codeword is a difficult and complex process when working with a non-systematic generator matrix. Multi-stage shift register technique is used in cyclic codes like RS codes to produce systematic generator matrix from a non-systematic one [5]. However, the technique does not work for AG codes since they are not cyclic, so another technique called Gauss-Jordan elimination could be used to convert the non-systematic generator matrix to a systematic one, keeping in mind that any interchange in columns while applying Gauss-Jordan elimination must be followed by same pattern on points [8, 13].

2.1.3 Designing AG Decoder

The traditional decoding technique for RS codes consists of two stages: the purpose of the first stage is to find the error locations while the second stage attempts to compute the error magnitudes for each of the found locations. AG codes follow the previously described technique [14].

In 1969, the BM algorithm [15] was introduced as a way to produce a shortest linear feedback shift register (LFSR) which yields a finite sequence of digits. By using the BM algorithm in 1988, Sakata was able to develop his algorithm which generates a set of minimal polynomials whose coefficients form a recursive relationship within a two-dimensional array of finite field elements [12].

Justesen et al. [16] were able to improve Sakata's algorithm in 1992. The aim of this improvement was to decrease the decoding complexity of AG codes by generating a set of error-locating polynomials (F) from a two-dimensional matrix containing syndrome values for AG codes. The decoding process starts with computing the elements in the two-dimensional syndromes array. Let us refer to the element location in the two dimensional array by $(S_{a,b})$ where a is the row number and b is the column number $(a, b < q - 1)$. The syndrome value is defined by [16]:

$$S_{a,b} = \sum_{i=1}^{n} r_i x_i^a y_i^b = \sum_{i=1}^{n} (c_i + e_i) x_i^a y_i^b = \sum_{i=1}^{n} e_i x_i^a y_i^b \tag{2.17}$$

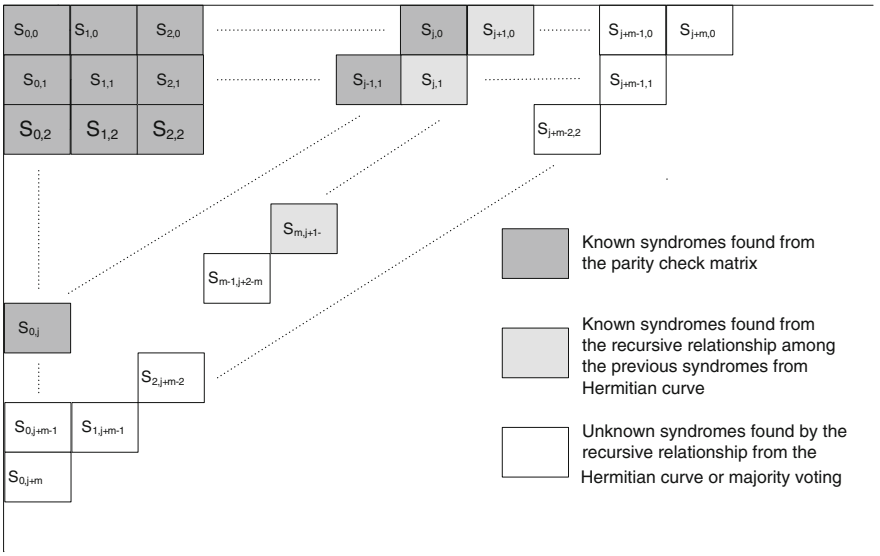

Fig. 2.1 General two-dimensional syndrome array

Let r_i be a received element within the received codeword r, c_i be a coded symbol, e_i the corresponding error magnitude in the i-th position, and (x_i, y_i) the i-th affine point, for $i \in I$, $I \subseteq \{1, 2, 3, ..., n\}$. The general two-dimensional syndrome array for the AG code constructed from the Hermitian curve defined by (2.1) is shown in Fig. 2.1 [5].

Sakata's algorithm makes use of this two-dimensional array by creating a set of error-locating polynomials (F) of the form [12]:

$$f^{(i)}(x, y) = \sum f_{k,l}^{(i)} x^k y^l \tag{2.18}$$

where the i-th polynomial in F is denoted by i, and the coefficients of the terms $x^k y^l$ in $f^{(i)}(x, y)$ is represented by $f^{(i)}(k, l)$. By reading every syndrome value in the two-dimensional array using the total graduate degree order $(<_T)$, all polynomials in F are updated in order to generate recursive relationships between known syndromes, up to the current syndrome by changing all the coefficients of every polynomial $f^{(i)}(x, y)$ [17]. However the generated recursive relationship needs to fulfill the following equation [5]:

$$\sum f_{k,l}^{(i)} S_{a-t_1^{(i)}+k, b-t_2^{(i)}+l} = 0 \tag{2.19}$$

where $f^{(i)}(x, y)$ is a polynomial in the set F and has x, y as variables of the leading term with $t_1^{(i)}$ and $t_2^{(i)}$ representing their powers, respectively. Using the total graduated degree ordering $(<_T)$ described previously, the syndromes in the two-

dimensional array are read as following: $S_{0,0}$, $S_{1,0}$, $S_{0,1}$, $S_{2,0}$, $S_{1,1}$, $S_{0,2}$, $S_{3,0}$, $S_{2,1}$, and so on until the last syndrome in the array which is $S_{15,15}$.

The nonnegative integer j defined by (2.4) plays an important role in articulating the syndromes. However, the syndromes are categorized into two types: known and unknown, where the known ones are from $S_{0,0}$ to $S_{m,j+1-m}$. To compute the syndromes $S_{0,0}$ to $S_{0,j}$ Eq. (2.17) is used. By substituting the curve equation in the equation representing the error-locating polynomial in (2.18), the following recursive relationship is formed to calculate more known syndromes $S_{j+1,0}$ to $S_{m,j+1-m}$ [12]:

$$\sum_{k,l} C_{k,l} S_{a-t_1^{(i)}+k,b-t_2^{(i)}+l} = 0 \tag{2.20}$$

$$C_{0,1} S_{a-m+0,b-0+1} + C_{0,m-1} S_{a-m+0,b-0+m-1} + C_{m,0} S_{a-m+m,b-0+0} = 0 \tag{2.21}$$

where the coefficient of $C(x, y)$ is $C_{k,l}$, and the powers of x and y for each term in $C(x, y)$ are k and l, respectively. This relationship could be simplified into the following equation as all coefficients of $C(x, y)$ are equal to one [5]:

$$S_{a,b} = S_{a-m,b+1} + S_{a-m,b+m-1} \tag{2.22}$$

Next is the time to update the set F by testing all polynomials ($f^{(i)}(x, y) \in F$) to see whether they satisfy (2.19). If they do, then none needs to be changed. Otherwise, if any of these polynomials do not satisfy (2.19) then this polynomial will be used in the updating process of the set F because it has a discrepancy d_f. This means that the polynomial at this stage is not ideal and must be changed so that it satisfies (2.19) after updating. The goal is to have a set of error-locating polynomials in F, and a polynomial is said to be error-locating if and only if it satisfies (2.19) [5, 8, 12].

The polynomials that do not satisfy (2.19) by having a nonzero discrepancy will be placed in a new set called auxiliary set (G). Also the point at which they were placed is stored (a_g, b_g). At this stage of the decoder, a new set $span(G)$ is generated by the union of all sets less than or equal to each $span(g^{(i)}(x, y))$ in G as in following Equation [5]:

$$span(G) = \sum_{i=1}^{\varphi} \{(k, l) \mid (k, l) \leq span(g^{(i)}(x, y))\} \tag{2.23}$$

where (k, l) are a pair of positive integers and φ is the number of polynomials in the set G. Span means that at the point (a, b) there is no polynomial with a leading term $x^{a_g - u_1^{(i)}} y^{b_g - u_2^{(i)}}$ that can satisfy (2.19). It is defined as [12]:

$$span(g^{(i)}(x, y)) = (a_g - u_1^{(i)}, b_g - u_2^{(i)}) \tag{2.24}$$

where $g^{(i)}(x, y)$ is a polynomial in the set G and has x, y as variables of the leading terms with $u_1^{(i)}$ and $u_2^{(i)}$ representing their degrees, respectively.

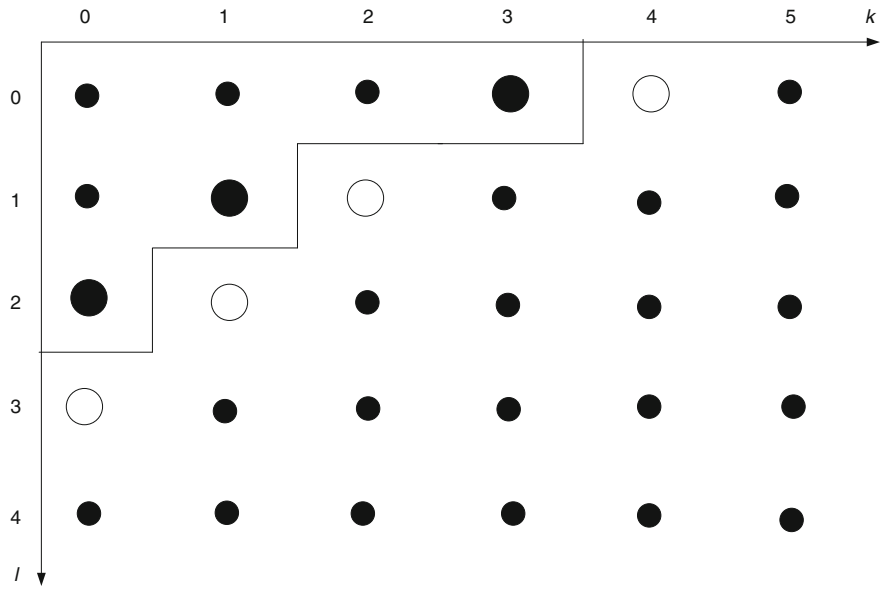

Fig. 2.2 Graphical representation of $span(G)$

The maximal point within $span(G)$ is defined as an interior corner while the minimal point outside $span(G)$ is called as exterior corner [5]. Both interior and exterior corners are defined with respect to partial ordering which is denoted by $(<)$ as mentioned earlier. However, the values of the exterior corners are the degrees of the polynomials in the set F and their number is the number of those polynomials.

Drawing $span(G)$ makes it much easier to find the interior and exterior corners. An example shows how drawing helps in finding out these corners. Assume $span(G) = \{(0, 0), (1, 0), (0, 1), (2, 0), (1, 1), (0, 2), (3, 0)\}$. From Fig. 2.2, the exterior corners are $(4, 0)$, $(2, 1)$, $(1, 2)$, and $(0, 3)$ as no other points outside $span(G)$ are less than them. In the same manner, it is shown that the interior corners are $(3, 0)$, $(1, 1)$, and $(0, 2)$ since these points are the greatest ones within $span(G)$. Exterior corners are marked with large white circles in the figure and the interior corners are marked with large black circles [5].

2.1.3.1 Updating the Sets F and G

The polynomials in the set F with a nonzero discrepancy ($d_f \neq 0$) are stored in a new set called F_N. The union of this set with the set G results a new set called G' which is the updated version of the set G ($G' = G \cup F_N$) [8]. Equation (2.23) is used to calculate the span of each polynomial in G', then Eq. (2.22) is used to find $span(G')$. The interior corners are found then using $span(G')$ so that any polynomial in the set G' with span not equal to any of the interior corner will be removed. In case two or

more polynomials in the set G' have span equal to any interior corner then any of those polynomials will be kept. The point in the two-dimensional syndrome array (a_g, b_g) where the discrepancies of remaining G' polynomials were nonzero and the discrepancy of the set G' polynomials d_g are stored [12]. G' at this stage is the final update of the set G which will be used for the next point in the two-dimensional syndrome array.

The exterior corners are found using $span(G')$ to update the set F. As mentioned earlier, the number of the exterior corners identifies the number of the polynomials in the updated set F'. Also their values are the powers of the leading terms of these polynomials [5]. The polynomials in the set F are updated using one of three cases for each of the exterior corners $(\varepsilon_1, \varepsilon_2)$, however these cases must be applied in the following order [8, 18]:

Case I If the difference set (F/F_N) has a polynomial $f^{(i)}(x, y)$ with $(t_1^{(i)}, t_2^{(i)}) = (\varepsilon_1, \varepsilon_2)$, then the new minimal polynomial $h^{(i)}(x, y) \in F'$ will be the same:

$$h^{(i)}(x, y) = f^{(i)}(x, y) \tag{2.25}$$

Case II If there is a polynomial $f^{(i)}(x, y) \in F_N$ with $(t_1^{(i)}, t_2^{(i)}) \leq (\varepsilon_1, \varepsilon_2)$ and $\varepsilon_1 > a$ or $\varepsilon_2 > b$, then the new minimal polynomial $h^{(i)}(x, y) \in F'$ is generated using:

$$h^{(i)}(x, y) = x^{\varepsilon_1 - t_1^{(i)}} y^{\varepsilon_2 - t_2^{(i)}} - f^{(i)}(x, y) \tag{2.26}$$

Case III If there is a polynomial $g^{(i)}(x, y) \in G$ having span greater than or equal to $(a-\varepsilon_1, b-\varepsilon_2)$ and a polynomial $f^{(i)}(x, y) \in F_N$ with $(t_1^{(i)}, t_2^{(i)}) \leq (\varepsilon_1, \varepsilon_2)$, then the new minimal polynomial $h^{(i)}(x, y) \in F'$ is generated using:

$$h^{(i)}(x, y) = x^{\varepsilon_1 - t_1^{(i)}} y^{\varepsilon_2 - t_2^{(i)}} f^{(i)}(x, y) - \frac{d_f}{d_g} x^{p_1} y^{p_2} g^{(i)}(x, y) \tag{2.27}$$

where $(p_1, p_2) = span(g^{(i)}(x, y)) - (a-\varepsilon_1, b-\varepsilon_2)$. Whenever, an update occurs to the set F, a new set denoted by Δ is developed. It will be used for the MV technique as part of the decoding procedure and also for termination of the decoding algorithm when required. It is defined as [8]:

$$\sum_{k=1}^{\lambda-1} \Delta_k \tag{2.28}$$

where λ represents the number of all polynomials in the set F. Further, Δ_k is defined by:

$$\Delta_k = \left\{ (k, l) \mid (k, l) \leq \left(t_1^{(k)} - 1, t_2^{(k+1)} - 1 \right) \right\} \tag{2.29}$$

where (k, l) are a pair of positive integers.

A major modification was introduced to the decoding algorithm which was concerned with adding a termination criteria for the algorithm when $|\Delta|$ becomes greater than the number of errors that the decoder can handle [5, 8]. In such case, the MV scheme will choose a false value for the unknown syndrome which will affect the accuracy of finding the remaining unknown syndromes resulting in inaccurate decoding.

After completing the two-dimensional syndrome array, all polynomials in the set F are said to be error-correcting polynomials which means when substituting the curve points into any of those polynomials, the error locations are the points that make the polynomial vanish [18].

A modified version of Sakata's algorithm is illustrated in the flow chart shown in Fig. 2.3. The best of our knowledge, this flow chart is the first published illustration in the literature.

2.1.3.2 Majority Voting

Sakata's algorithm uses the technique of substituting the curve Eq. (2.1) into (2.19) to come up with a recursive relationship among the previous syndromes to find the unknown syndromes of the type $S_{a,b}$ where $a \leq m$. This can be true if and only if all previous syndromes are known. If any of those previous syndromes are unknown, then the MV technique is used to compute the unknown syndromes of the type $S_{a,b}$ where $a < m$ [5, 12].

The following example will help clarify the idea. For an algebraic geometric code constructed from a Hermitian curve of the form given in (2.1) with degree $m = 5$, Eq. (2.22) can be used to compute the syndrome $S_{8,1}$:

$$S_{8,1} = S_{8-5,1+1} + S_{8-5,5-1} \tag{2.30}$$

$$= S_{3,2} + S_{3,4} \tag{2.31}$$

This only holds if both $S_{3,2}$ and $S_{3,4}$ are known. Otherwise, MV technique is used to find the unknown ones.

In 1993 Feng and Rao [19] introduced the MV scheme which Sakata et al. used later in 1995 [17] to design a hard-decision decoding technique for AG codes. For an unknown syndrome of the type $S_{a,b}$ where $a < m$, any minimal polynomial $f^{(i)}(x, y) \in F$ will be used to find a candidate syndrome value. It turns out that there are four possible scenarios to be encountered depending on some conditions which will be explained in detail below[17].

Scenario one: The candidate syndrome value is v_i if $a = t_1^{(i)}$ and $b = t_2^{(i)}$ can be calculated by using the following equation which is derived from (2.19):

$$\sum_{(k,l)\leq_T \left(t_1^{(i)}, t_2^{(i)}\right)} f_{k,l}^{(i)} S_{k+a-t_1^{(i)}, l+b-t_2^{(i)}} = -v_i \tag{2.32}$$

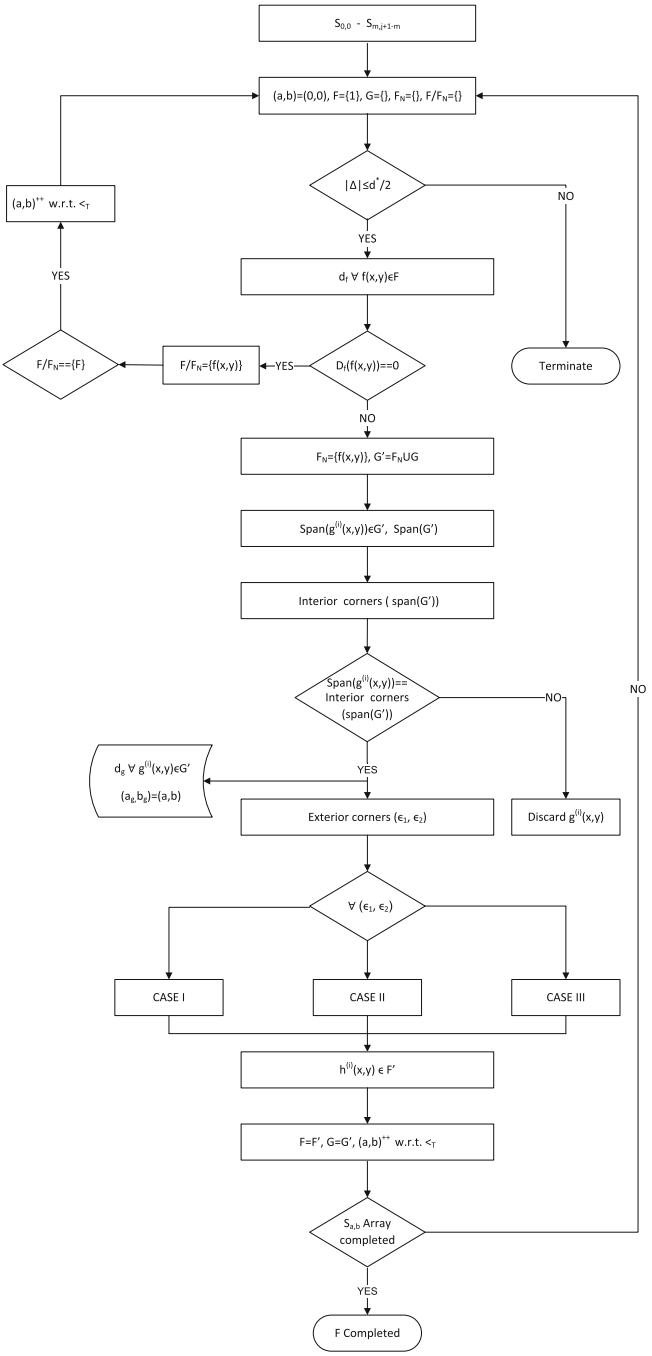

Fig. 2.3 Flow chart of modified version of Sakata's algrithm

Scenario two: The candidate syndrome value is w_i if $a+m=t_1^{(i)}$ and $b-t_2^{(i)}=m-1$ can be calculated by using the following equation:

$$\sum_{(k,l)<_T\left(t_1^{(i)},t_2^{(i)}\right)} f_{k,l}^{(i)} S_{k+a+m-t_1^{(i)},l+b-m+1-t_2^{(i)}} - S_{a,b-m+2} = -w_i \tag{2.33}$$

Scenario three: If both scenarios one and two are fulfilled which means (2.32) and (2.33) are satisfied, then there will be two candidate values v_i and w_i for the syndrome from this minimal polynomial.

Scenario four: If none of the above three scenarios are fulfilled, then the chosen minimal polynomial $f^{(i)}(x, y) \in F$ is not capable of finding a candidate syndrome value, so a different minimal polynomial $f^{(i)}(x, y) \in F$ will be considered.

The next step in calculating the MV is to generate two new sets:

$$\begin{aligned} K_1 &= \{(k,l) \mid 0 \le k \le a \wedge 0 \le l \le b\} \\ K_2 &= \{(k,l) \mid 0 \le k < m \wedge 0 \le l \le b-m+1\} \end{aligned} \tag{2.34}$$

where (k, l) are a pair of positive integers. A set $K = K_1 \cup K_2$ is computed also. The MV decision is made based on the number of elements in the set K_j which is found for each candidate syndrome value $\delta_1, \delta_2, \delta_3, \ldots$ as [17]:

$$K_j = \left(\bigcup_{v_i=\delta_j} A_i \cup \bigcup_{w_i=\delta_j} B_i \right) \Big/ \Delta \tag{2.35}$$

where A_i and B_i are defined by:

$$\begin{aligned} A_i &= \left\{(k,l) \in K \mid k + t_1^{(i)} \le a \wedge l + t_2^{(i)} \le b\right\} \\ B_i &= \left\{(k,l) \in K \mid k + t_1^{(i)} \le a+m \wedge l + t_2^{(i)} \le b-m+1\right\} \end{aligned} \tag{2.36}$$

2.1.3.3 Error Magnitudes

To find the error magnitude for AG codes (generated from Hermitain curves), the points on the curve are categorized into four types. The magnitude of the error will be calculated based on the error location on the curve which means it does matter where the error occurs in order to find its magnitude [8].

The following four categories of points on the curve will be useful in the method described below. The method depends on calculations of a one-dimensional inverse discrete Fourier transform (IDFT), knowledge of the unknown syndromes up to syndrome $S_{q-1,q-1}$, and the curve properties [17, 20].

Category one: For error occurring at the point where both coordinates x and y are zeros, i.e., $P_{(0,0)}$. The error value is found by subtracting the error values of errors which occurred at all other points types $P_{(x,0)}$, $P_{(0,y)}$, and $P_{(x,y)}$ as following:

$$\begin{aligned} \sum_{p_i \in P_{(x,y)}} e_i &= S_{q-1,q-1} \\ \sum_{p_i \in P_{(0,y)}} e_i &= S_{0,q-1} - S_{q-1,q-1} \\ \sum_{p_i \in P_{(x,0)}} e_i &= S_{q-1,0} - S_{q-1,q-1} \end{aligned} \tag{2.37}$$

This leads to:

$$\begin{aligned} e_i &= \sum_i e_i - \sum_{p_i \in P_{(x,y)}} e_i - \sum_{p_i \in P_{(0,y)}} e_i - \sum_{p_i \in P_{(x,0)}} e_i \\ &= S_{0,0} - S_{q-1,q-1} - (S_{0,q-1} - S_{q-1,q-1}) - (S_{q-1,0} - S_{q-1,q-1}) \end{aligned} \tag{2.38}$$

However, for the codes constructed from the curves over a finite field of characteristic two, Eq. (2.38) can be simplified to:

$$e_i = S_{0,0} + S_{q-1,0} + S_{q-1,q-1} + S_{0,q-1} \tag{2.39}$$

Category two: For all errors occurring at the points of zero x-coordinate and nonzero y-coordinate, i.e., $P_{(0,y)}$, the following mapping is defined:

$$m \rightarrow \begin{cases} \alpha^m & \text{for } 0 \leq m \leq q-2 \\ 0 & \text{for } m = q-1 \end{cases} \tag{2.40}$$

and the one-dimensional IDFT equation is:

$$E_n = \sum_{i=0}^{q-2} S_{0,q-1-i} \alpha^{ni} \tag{2.41}$$

where α is the primitive element of the finite field and E_n is the summation of all error values occurred at the points of nonzero y-coordinate α^n. Luckily, Hermitian curves (the focus here) have a property that whenever there is a point on the curve of zero x-coordinate and nonzero y-coordinate, there will be no points on the curve with the same y-coordinate value with nonzero x-coordinate (α^m, α^n). Which means that E_n is in fact the error magnitude of the error at the point $P_{(0,y)} = (0, \alpha^n)$.

Category three: For all errors occurring at the points of nonzero x-coordinate and zero y-coordinate, i.e., $P_{(x,0)}$, the same mapping (2.40) as above takes place and the one-dimensional IDFT relation is:

$$E_m = \sum_{i=0}^{q-2} S_{q-1-i,0}\alpha^{mi} \tag{2.42}$$

where α is the primitive element of the finite field and E_m is the summation of all error values happening at the points of nonzero x-coordinate α^m. The property of Hermitian curves mentioned above still applies which says there are no points on the curve with the same x-coordinate value α^m and nonzero y-coordinate. Hence, E_m is in fact the error magnitude of the error at the point $P_{(x,0)} = (\alpha^m, 0)$.

Category four: A two-dimensional IDFT is used for errors occurring at the points of nonzero x-coordinate and nonzero y-coordinate, $P_{(x,y)}$. The error magnitude of the error at any point $P_i = (x, y)$ is given by:

$$e_i = \sum_{a=0}^{q-2}\sum_{b=0}^{q-2} S_{a,b}x_i^{-a}y_i^{-b} \tag{2.43}$$

where e_i is the error magnitude of the error that happened at the point P_i, and q is the size of the finite field. However, before Sakata et al. started this method in 1995, which was later improved by Liu [20] in 1999, a very lengthy and complex method was found by solving Eq. (2.17).

2.1.4 Complete Hard-Decision Decoding Algorithm for AG Codes Constructed From Hermitian Curves

In this section, we describe the details of the decoding algorithm used to decode AG codes constructed from Hermitian curves. It is used for iterative decoding [4, 5, 8] later in this book.

Step 1: Known syndromes computation:

a. The known syndromes $S_{0,0}, \ldots, S_{0,j}$ can be found by applying Eq. (2.17).
b. The known syndromes $S_{j+1,0}, \ldots, S_{m,j-m+1}$ can be found using Eq. (2.22).

Step 2: Finding the error location:
The known syndromes and some of the unknown syndromes up to $S_{0,j+m}$ are needed to find the error locations.

a. Run Sakata's algorithm with known syndromes (found in step 1) as input; some unknown syndromes are found using (2.22) when syndrome is of the form $S_{a,b}$ for $b \geq m-1$.
b. Run Sakata's algorithm with unknown syndromes (found in step 2-a) as input; when having a syndrome of the form $S_{a,b}$ for $a \geq m$, then (2.22) is used to compute the value of the unknown syndrome or MV scheme is used if the syndrome has the form $S_{a,b}$ for $a < m$.

c. Run Sakata's algorithm with unknown syndromes (found step 2-b) as input and find more unknown syndromes using (2.22) when syndrome is of the form $S_{a,b}$ for $b \geq m - 1$.
d. Substitute the points on the curve into any of the minimal (error-locating) polynomials in set F to find its roots as these roots are the error locations.

Step 3: Finding magnitudes of errors:
The unknown syndromes from $S_{j+1+m,0}$ up to the last syndrome of the two-dimensional syndrome array $S_{q-1,q-1}$ are needed to compute the error magnitudes.

a. Equation (2.22) is used to find the value of the unknown syndrome if it is of the form $S_{a,b}$ for $a \geq m$.
b. If the unknown syndrome is of the form $S_{a,b}$ for $a < m$, then to compute its value, a recursive relationship between the syndromes should be formed by substituting the last minimal polynomial in the set F in Eq. (2.19).
c. Find the error values using IDFT:

- When the error location is at the origin point $P_{x,y} = (0, 0)$, then Eq. (2.39) is used to find the error magnitude.
- When the error is located at a point with zero x-coordinate and nonzero y-coordinate $P_{x,y} = (0, y)$, then Eq. (2.41) is used to find the error magnitude.
- When the error is located at a point with nonzero x-coordinate and zero y-coordinate $P_{x,y} = (x, 0)$, then Eq. (2.42) is used to find the error magnitude.
- When the error is located at a point with nonzero x-coordinate and nonzero y-coordinate $P_{x,y} = (x, y)$, then Eq. (2.43) is used to find the error magnitude.

Step 4: Error correction:
To correct the errors in terms of extracting the original message, the error values found in step 3 at the positions found in step 2 are added into the received codeword to give the decoded codeword. Then the original message is the first k symbols from the decoded codeword as the code is systematic.

2.2 Turbo Codes

Turbo coding was a breakthrough in channel coding introduced in 1993 by a group of French researchers [21, 22] as a new class of error correction codes with a relevant iterative decoding method. Turbo coding was not just a new set of codes but a new way of thinking about channel coding. These codes showed performance close to Shannon's capacity limit [21]. This represented a significant gain in power efficiency over other coding techniques known at that time.

The operation of a turbo codec relies on some basic ideas: using uncorrelated inputs, divide and conquer, and processing information iteratively. The information to be transmitted is stored in a memory in order to be scrambled to produce two

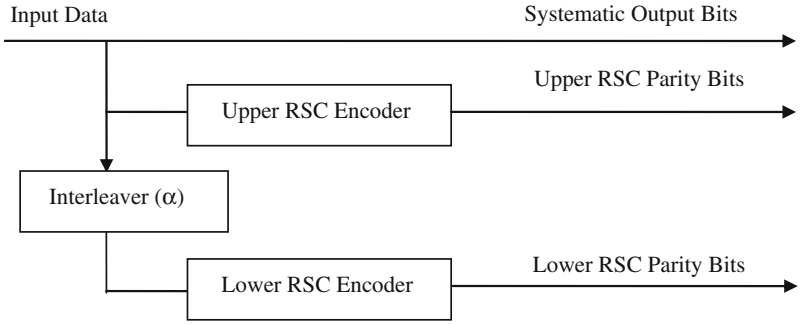

Fig. 2.4 Turbo encoder

uncorrelated sequences that are then encoded and transmitted. This idea is the key to the incomparable performance of turbo codes [23].

Since TCs were introduced, they have been useful for low-power applications such as satellite, deep-space communications, and for interference limited applications such as third generation (3G) cellular and personal communication services. Even though TCs have been a "hot topic" in the research literature over the past decade, there is still a relative lack of basic and fundamental papers serving as a starting point for researchers in this field [24]. The following sections in this chapter will briefly describe the main three components of a turbo codec (turbo encoder, interleaver, and turbo decoder).

2.2.1 Turbo Encoder

The basic turbo code encoder is produced using parallel concatenation of two identical recursive systematic convolutional (RSC) encoders separated by arbitrary interleaver (other interleavers could also be used such as block interleaver) [21, 25] as shown in Fig. 2.4.

This way of constructing an encoder is called parallel concatenation because the two encoders operate on the same input bits, rather than one encoding the output of the other. As a result, TCs are called parallel concatenated convolutional codes (PCCC) [26].

Both encoders have the same rate ($r = 1/2$), the upper encoder receiving data directly while the lower one receives it after being randomly interleaved by a permutation function α which maps bits in position i to position $\alpha(i)$. It is important to note that this interleaver α works in a block-wise manner, interleaving L bits at a time. Hence, TCs are actually block codes [25]. As both encoders receive the same input sequence in permuted fashion then only one of the systematic outputs needs to be transmitted. In most turbo encoders, the systematic output of the upper encoder is sent along with the parity bits of both of them. The overall rate of a

TC consisting of parallel concatenation of two systematic codes with rate ($r = 1/2$) is ($r = 1/3$). However, this rate can be increased if a subset of the parity bits is stopped from being transmitted by a process called puncturing. The code rate of a TC is increased to ($r = 1/2$) if the odd indexed parity bits and all systematic bits from the upper encoder are transmitted along with the even indexed parity bits from the other encoder [24].

2.2.2 Interleaver

The interleaver in turbo coding is a pseudorandom block scrambler which permutes N input bits with no repetitions by reading it into the interleaver and reading it out pseudorandomly [23, 25]. The interleaver has two main roles in TC: converting the small memory convolutional codes into long block codes, and decorrelating the inputs to both decoders so that an iterative sub-optimal decoding algorithm based on information exchange between the two decoders can be applied. This role of the interleaver makes it necessary that the same interleaving pattern should be available at the decoding side [21, 22]. If the input sequences to the two decoders are decorrelated, then there is a high possibility that after correction some of the errors in one decoder and some of the remaining errors become correctable in the second decoder [25].

2.2.3 Turbo Decoder

The TC decoder is constructed in a similar way as the encoder. Two simple soft-input soft-output (SISO) decoders are interconnected to each other in a serial concatenation. An interleaver is installed between the two decoders to spread out error bursts coming from the output of first decoder [21].

TCs can be decoded by maximum a posteriori (MAP) or maximum likelihood (ML) decoding methods. These decoders could be implemented only for small size interleavers as they are too complex for medium and large interleaver sizes [26]. The realistic value of TCs lies in the availability of a simple sub-optimal decoding algorithm [21, 26].

The idea behind turbo decoding is improving the reliability of the second decoder output by feeding it with extrinsic information that has been extracted out of the first decoder output. Then the reliability of the first decoder's output is improved by feeding the first decoder with extrinsic information extracted from the second decoder's output. This process will keep iterating until no further improvement can be made on the performance of the turbo decoder [24].

2.3 Block Turbo Codes

In 1994 Ramesh Pyndiah introduced BTCs as an alternative to classical convolutional TCs which were introduced a year before for applications requiring either high code rates ($R > 0.8$), very low error floors, or low complexity decoders that operate at several hundreds of megabits per second or higher [27].

BTCs are constructed as the data to be encoded is set in an l-dimensional hypercube with dimensional lengths denoted by $(k_1, k_2, \ldots, k_l)$. Here all the dimensional sub-codes are encoded in the systematic linear block code (n_i, k_i, d_{mini}), where n_i represents the length of the code, k_i is the length of the information bit, d_{mini} is the minimum Hamming distance, and $r_i = k_i/n_i$ the code rate of the i-th dimensional sub-code. As a result for the l-dimensional BTC, the codeword length is $\prod_{i=1}^{l} n_i$; the information bit length is $\prod_{i=1}^{l} k_i$; the minimum Hamming distance is $\prod_{i=1}^{l} d_{mini}$; and the code rate is $\prod_{i=1}^{l} r_i$. Note that a higher dimensional number of the BTC implies a more complex implementation so the two-dimensional BTC seems to be the right choice for communication systems because of its relatively simple implementation and suitable structure for high code rate codes [28].

The RS code or Bose-Chaudhuri-Hocguenghem (BCH) code can be chosen as the component code of a two-dimensional BTC. The RS code has better error correction performance but due to its very high decoding complexity, the BCH code is usually preferred for practical applications [23].

To encode a two-dimensional BTC whose component code is a BCH code, first the $k_1 \times k_2$ information bits are set into a matrix of k_2 rows and k_1 columns. Then the k_2 rows are horizontally or row-wise encoded by applying $BCH(n_1, k_1, d_{min1})$ and k_1 columns are vertically or column-wise encoded by applying $BCH(n_2, k_2, d_{min2})$ as shown in Fig. 2.5 [25].

In addition, a row/column interleaver is used in between the two BCH encoders to guarantee the information bit that is horizontally encoded in the first BCH encoder can be vertically encoded in the second BCH encoder. One can see that this encoding technique is identical to encoding a BCH serial concatenated code in which the same interleaver used. Encoding with this technique leads to a BTC with the following parameters: $n = n_1 \times n_2$, $k = k_1 \times k_2$, and $d_{min} = d_{min1} \times d_{min2}$.

Concerning the decoding process, let us consider the decoding of binary linear block code $c(n, k, d_{min})$. While for high rate block code whose codeword length is too long, ML decoding requires very large code numbers and the complexity of the decoding algorithm increases exponentially. Therefore, a decoding technique with much lower complexity and small degradation in performance for the linear block code was introduced by Chase in 1972 and used by Pyndiah in 1994 [27]. It should be noted that the previous technique is also suitable for decoding non-binary codes like RS codes.

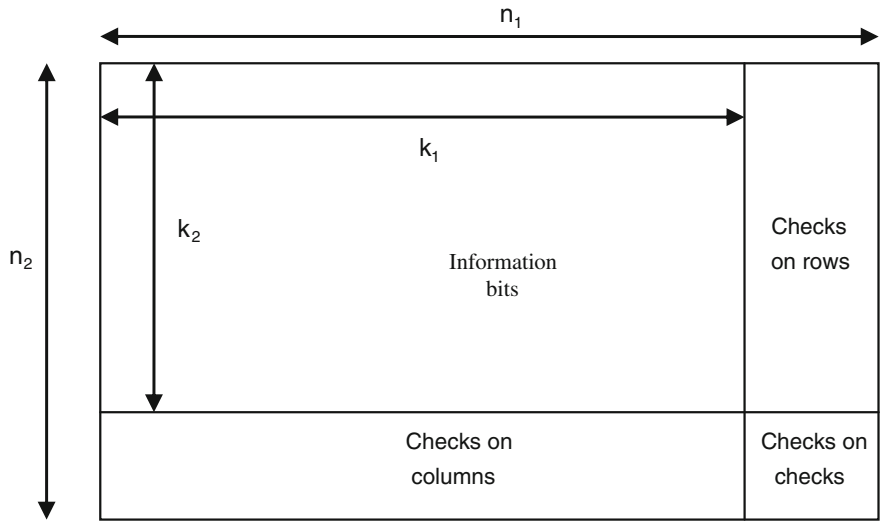

Fig. 2.5 The encoding structural diagram of the two-dimensional BTC

2.4 Summary

In this chapter, an overview of the concepts of AG code construction, encoding, and decoding techniques has been presented in detail which forms a foundation for understanding the subsequent chapters.

AG code construction sets out the code parameters such as the message length, codeword length, minimum Hamming distance, and the capabilities of code in locating and correcting errors. The encoding part was mainly composed of generating a non-systematic generator matrix and converting that into a systematic one using Gauss-Jordan elimination technique. For decoding, a full description was given of Sakata's algorithm and the MV technique was explained as well. Finding the magnitudes of errors depending on their locations was also explained.

Also in this chapter, the basics of TC were reviewed through a brief description of the main three components of TC (turbo encoder, interleaver, and turbo decoder). BTCs were introduced briefly as a prelude to more detailed explanations to follow in the next chapters.

References

1. Goppa VD (1981) Codes on algebraic curves. Soviet Math Dokl 24:75–91
2. Ozbudak F, Stichtenoth H (1999) Constructing codes from algebraic curves. IEEE Trans Inf Theory 45(7):2502–2505. doi:10.1109/18.796391
3. Tsfasman MA, Vladut SG, Zink T (1982) Modular curves, shimura curves and goppa codes, better than Varshamov-Gilbert bound. Math Nachtrichten 109:21–28

4. Justesen J, Larsen K, Jensen H, Havemose A, Hoholdt T (1989) Construction and decoding of a class of algebraic geometry codes. IEEE Trans Inf Theory 35(4):811–821. doi:10.1109/18.32157
5. Carrasco RA, Johnston M (2008) Non-binary error control coding for wireless communication and data storage. Wiley, New York. doi:10.1002/9780470740415.fmatter. http://dx.doi.org/10.1002/9780470740415.fmatter
6. Kirwan F (1992) Complex algebraic curves. London mathematical society student texts, vol. 23. Cambridge University Press, Cambridge. doi:10.2277/0521423538
7. Walker RJ (1978) Algebraic curves. Princeton mathematical series, vol. 13. Springer-Verlag, New York
8. Johnston M, Carrasco RA (2005) Construction and performance of algebraic-geometric codes over awgn and fading channels. IEE Proc Commun 152(5):713–722. doi:10.1049/ip-com:20045153
9. Blake I, Heegard C, Hoholdt T, Wei V (1998) Algebraic-geometry codes. IEEE Trans Inf Theory 44(6):2596–2618. doi:10.1109/18.720550
10. Pretzel O (1998) Codes and algebraic curves. Oxford University Press, New York
11. Wicker SB (1995) Error control systems for digital communication and storage. Prentice-Hall, Upper Saddle River
12. Sakata S (1988) Finding a minimal set of linear recurring relations capable of generating a given finite two-dimensional array. J Symbolic Comput 5(3):321–337. doi:10.1016/S0747-7171(88)80033-6. http://www.sciencedirect.com/science/article/pii/S0747717188800336
13. Atkinson K (1989) An introduction to numerical analysis, 2nd edn. Wiley, New York. http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0471624896
14. Reed IS, Solomon G (1960) Polynomial codes over certain finite fields. J Soc Ind Appl Math 8(2):300–304
15. Massey J (1969) Shift-register synthesis and bch decoding. IEEE Trans Inf Theory 15(1):122–127. doi:10.1109/TIT.1969.1054260
16. Justesen J, Larsen K, Jensen H, Hoholdt T (1992) Fast decoding of codes from algebraic plane curves. IEEE Trans Inf Theory 38(1):111–119. doi:10.1109/18.108255
17. Sakata S, Justesen J, Madelung Y, Jensen H, Hoholdt T (1995) Fast decoding of algebraic-geometric codes up to the designed minimum distance. IEEE Trans Inf Theory 41(6):1672–1677. doi:10.1109/18.476240
18. Saints K, Heegard C (1995) Algebraic-geometric codes and multidimensional cyclic codes: a unified theory and algorithms for decoding using Grobner bases. IEEE Trans Inf Theory 41(6):1733–1751. doi:10.1109/18.476246
19. Feng GL, Rao TRN (1993) Decoding algebraic-geometric codes up to the designed minimum distance. IEEE Trans Inf Theory 39(1):37–45. doi:10.1109/18.179340
20. Liu CW (1999) Determination of error values for decoding Hermitian codes with the inverse affine Fourier transform. IEICE Trans Fundam Electron Commun Comput Sci 82(10):2302–2305. http://ci.nii.ac.jp/naid/110003208168/en/
21. Berrou C, Glavieux A, Thitimajshima P (1993) Near shannon limit error-correcting coding and decoding: turbo-codes. 1. In: IEEE International Conference on Communications (ICC'93), vol. 2. pp. 1064–1070. doi:10.1109/ICC.1993.397441
22. Berrou C, Glavieux A (1996) Near optimum error correcting coding and decoding: turbo-codes. IEEE Trans Commun 44(10):1261–1271. doi:10.1109/26.539767
23. Sklar B (1988) Digital communications: fundamentals and applications. Prentice-Hall, Upper Saddle River
24. Valenti MC (1998) Turbo codes and iterative processing. In: Proceedings IEEE New Zealand Wireless Communications Symposium, pp. 216–219
25. Vucetic B, Yuan J (2000) Turbo codes: principles and applications. Kluwer Academic, Boston. http://www.loc.gov/catdir/enhancements/fy0820/00033104-t.html
26. Benedetto S, Montorsi G (1996) Unveiling turbo codes: some results on parallel concatenated coding schemes. IEEE Trans Inf Theory 42(2):409–428. doi:10.1109/18.485713

27. Pyndiah R, Glavieux A, Picart A, Jacq S (1994) Near optimum decoding of product codes. In: Proceedings of IEEE GLOBECOM '94, pp. 339–343. doi:10.1109/GLOCOM.1994.513494
28. Pyndiah RM (1998) Near-optimum decoding of product codes: block turbo codes. IEEE Trans Commun 46(8):1003–1010. doi:10.1109/26.705396

http://www.springer.com/978-3-319-08292-9

Forward Error Correction Based On Algebraic-Geometric Theory

Alzubi, J.; Alzubi, O.; M. Chen, Th.

2014, XII, 70 p. 33 illus., 20 illus. in color., Softcover
ISBN: 978-3-319-08292-9