IET Communications

Research Article

Progressive algebraic Chase decoding algorithms for Reed–Solomon codes

ISSN 1751-8628
Received on 7th September 2015
Revised on 13th December 2015
Accepted on 27th February 2016
doi: 10.1049/iet-com.2015.0873
www.ietdl.org

Jiancheng Zhao[1], *Li Chen*[1] ✉, *Xiao Ma*[2], *Martin Johnston*[3]

[1]*School of Electronics and Information Technology, Sun Yat-sen University, Guangzhou 510006, People's Republic of China*
[2]*School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, People's Republic of China*
[3]*School of Electrical and Electronic Engineering, Newcastle University, Newcastle-upon-Tyne NE1 7RU, UK*
✉ *E-mail: chenli55@mail.sysu.edu.cn*

Abstract: This study proposes a progressive algebraic Chase decoding (PACD) algorithm for Reed–Solomon (RS) codes. On the basis of the received information, 2^{η} ($\eta > 0$) interpolation test-vectors are constructed for the interpolation-based algebraic Chase decoding. A test-vector reliability function is defined to assess their potential for yielding the intended message. The algebraic Chase decoding will then be performed progressively granting priority to decode the test-vectors that are more likely to yield the message, and is then terminated once it is found. Consequently, the decoding complexity can be adapted to the quality of the received information. An enhanced-PACD (E-PACD) algorithm is further proposed by coupling the PACD algorithm with the adaptive belief propagation (ABP) decoding. The ABP decoding generates new test-vectors for the PACD algorithm by enhancing the received information. It improves the Chase decoding performance without increasing the decoding complexity exponentially. It is shown that the E-PACD algorithm's complexity can be significantly reduced by utilising the existing interpolation information of the previous Chase decodings'. Our performance evaluations show that the two proposed decoders outperform a number of existing algebraic decoding approaches. Complexity and memory analyses of the PACD algorithm are also presented, demonstrating that this is an efficient RS decoding strategy.

1 Introduction

Reed–Solomon (RS) codes [1] are widely used in storage and communication systems. The conventional decoding algorithms [2, 3] produce a unique decoded message, with error-correction capability limited to the half of the code's minimum Hamming distance. The generalised minimum-distance decoding algorithm [4] is an improved unique decoding approach. The Guruswami–Sudan (GS) [5, 6] algebraic list decoding algorithm was later proposed to correct errors beyond the half distance bound by performing a curve-fitting decoding process that contains two major steps: interpolation and factorisation. The Koetter–Vardy (KV) soft-decision list decoding algorithm [7] was later proposed by introducing a reliability transform front-end. It significantly outperforms the GS and the unique decoding algorithms with a polynomial-time complexity.

However, the complexity of the algebraic list decoding algorithms is orders of magnitude higher than that of the unique decoding algorithms. This is mainly caused by the interpolation, which is an iterative polynomial construction process [8, 9]. Addressing this issue, a number of complexity reducing interpolation approaches have been proposed. During interpolation, redundant polynomials can be identified by assessing their leading order, which will be eliminated removing the unnecessary computation [8]. Interpolation complexity can also be reduced by utilising the Berlekamp–Massey decoding output, which leads to a reduced interpolation multiplicity [10]. The re-encoding approach [11] reduces the complexity by transforming the interpolation points to have zero y-coordinates. Consequently, a large part of iterative polynomial construction can be replaced by the polynomial initialisation. The low-complexity Chase (LCC) decoding [12, 13] reduces the complexity by exploiting the similarity among several interpolation test-vectors. It has been shown that the LCC algorithm can outperform the KV algorithm with less computational cost. However, its complexity can increase exponentially by increasing the number of unreliable symbols. By arranging the test-vectors in a way such that the adjacent test-vectors have only one different point, the backward–forward (BF) interpolation can be applied, resulting in a hardware friendly BF-LCC decoder [14, 15]. Moreover, a tree-based Chase decoding approach that better defines the test-vector set was recently reported in [16]. To adapt the algebraic decoding computation to the quality of the received information, progressive algebraic soft decoding was recently proposed by Chen *et al.* [17]. By progressively enlarging the designed factorisation output list size (OLS), it uses the least necessary algebraic decoding effort in finding the intended message. In good channel conditions, most of the decoding events can deliver the intended message with a small OLS value. As a result, the average complexity of multiple decoding events can be reduced. Similarly, an interpolation algorithm that adjusts its computation to the number of instantaneous errors was proposed in [18].

To reduce the average decoding complexity while maintaining a high RS decoding performance, this paper proposes the progressive algebraic Chase decoding (PACD) algorithm which integrates the methodologies of progressive and LCC decodings. A reliability function is defined to assess each test-vector's potential for yielding the intended message. The test-vector with a higher potential is decoded before the less potential one, and the decoding will be terminated once the intended message is yielded. This is validated by the maximum-likelihood (ML) criterion [19]. To improve the decoding performance without incurring an exponentially increased complexity, an enhanced-PACD (E-PACD) algorithm is further proposed. Instead of increasing the number of unreliable symbols for forming more test-vectors, it couples the PACD algorithm with the adaptive belief propagation (ABP) decoding [20, 21]. The ABP decoding enhances the received information and generates new interpolation test-vectors for the PACD algorithm. By eliminating the repeated test-vectors and using the stored interpolation information, redundant computation of the later PACD attempts can be avoided. It attains a significant performance improvement without incurring an exponentially increased computational cost. Our performance evaluation demonstrates the two proposed decoders have superior error-correction performance compared with a number of existing algebraic decoding algorithms. The decoding complexity of the PACD algorithm is analysed and validated by numerical results, highlighting its channel dependence feature. It also shows that incorporating progressive decoding enables the PACD algorithm to be simpler than its predecessors, i.e. the LCC and the BF-LCC algorithms. The memory requirement for the PACD algorithm is also analysed.

The rest of this paper is organised as follows: Section 2 presents the preliminaries of this paper. Section 3 presents the PACD algorithm in detail. Section 4 introduces the E-PACD algorithm. Section 5 presents the simulation results showing the error-correction performance of the PACD and the E-PACD algorithms. Sections 6 and 7 analyse the decoding complexity and memory requirement of the PACD algorithm, respectively. Finally, Section 8 concludes this paper.

2 Preliminaries

Let $\mathbb{F}_q = \{0, 1, \alpha, \ldots, \alpha^{q-2}\}$ denote the finite field of size q, where α is the primitive element. In this paper, it is assumed that $\mathbb{F}_q$ is an extension field of $\mathbb{F}_2$ and $q = 2^\rho$, where ρ is a positive integer. For convenience, $\mathbb{F}_q$ can also be denoted as $\mathbb{F}_q = \{0, 1, 2, \ldots, q-1\}$. Let $\mathbb{F}_q[x]$ and $\mathbb{F}_q[x, y]$ denote the univariate and the bivariate polynomial rings defined over $\mathbb{F}_q$, respectively. We consider an (n, k) RS code, where n $(n = q-1)$ and k $(k < n)$ are the length and dimension of the code, respectively. Given a message vector $\overline{\boldsymbol{M}} = (M_0, M_1, \ldots, M_{k-1}) \in \mathbb{F}_q^k$, the message polynomial can be written as

$$M(x) = \sum_{u=0}^{k-1} M_u x^u. \tag{1}$$

The RS codeword can be generated by

$$\overline{\boldsymbol{C}} = (C_0, C_1, \ldots, C_{n-1}) = (M(x_0), M(x_1), \ldots, M(x_{n-1})), \tag{2}$$

where $\{x_0, x_1, \ldots, x_{n-1}\} \in \mathbb{F}_q \backslash \{0\}$ and $\overline{\boldsymbol{C}} \in \mathbb{F}_q^n$.

The parity-check matrix of the RS code is defined as

$$\boldsymbol{H} = \begin{pmatrix} 1 & \alpha & \cdots & \alpha^{n-1} \\ 1 & \alpha^2 & \cdots & \alpha^{2(n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-k} & \cdots & \alpha^{(n-k)(n-1)} \end{pmatrix}. \tag{3}$$

An RS codeword $\overline{\boldsymbol{C}}$ satisfies $\overline{\boldsymbol{C}} \cdot \boldsymbol{H}^{\mathrm{T}} = \boldsymbol{0}$, where $\boldsymbol{0}$ is the all-zero vector. Let $\sigma(x)$ be a primitive polynomial of $\mathbb{F}_q$ and $\sigma(x) \in \mathbb{F}_2[x]$. Its companion matrix $\boldsymbol{A}$ is a $\rho \times \rho$ binary matrix. For a finite field element α^φ $(\varphi = 0, 1, \ldots, q-2)$, the mapping $\alpha^\varphi \mapsto \boldsymbol{A}^\varphi$ can be performed to obtain its binary image. Therefore, the binary image of matrix $\boldsymbol{H}$ can be obtained by replacing its entries α^φ with $\boldsymbol{A}^\varphi$. Let $\boldsymbol{H}_{\mathrm{b}}$ denote such a binary parity-check matrix with size $(N-K) \times N$, where $N = n\rho$ and $K = k\rho$. Given $\overline{\boldsymbol{c}} = (c_0, c_1, \ldots, c_{N-1}) \in \mathbb{F}_2^N$ as the binary representation of an RS codeword $\overline{\boldsymbol{C}}$, we have $\overline{\boldsymbol{c}} \cdot \boldsymbol{H}_{\mathrm{b}}^{\mathrm{T}} = \boldsymbol{0}$.

The PACD algorithm organises monomials $x^a y^b$ using the $(1, -1)$-lexicographic order (ord), where $(a, b) \in \mathbb{N}^2$ and $\mathbb{N}$ denotes the set of non-negative integers. For a monomial $x^a y^b$, its $(1, -1)$-weighted degree is defined as $\deg_{1,-1}(x^a y^b) = 1 \cdot a + (-1) \cdot b$. The $(1, -1)$-lexicographic order implies $\mathrm{ord}(x^{a_1} y^{b_1}) < \mathrm{ord}(x^{a_2} y^{b_2})$, if $\deg_{1,-1}(x^{a_1} y^{b_1}) < \deg_{1,-1}(x^{a_2} y^{b_2})$ or $\deg_{1,-1}(x^{a_1} y^{b_1}) = \deg_{1,-1}(x^{a_2} y^{b_2})$ and $a_1 > a_2$. Given a polynomial $Q = \sum_{a,b} Q_{ab} x^a y^b \in \mathbb{F}_q[x, y]$, if $x^{a'} y^{b'}$ is the leading monomial with coefficient $Q_{a'b'} \neq 0$, its leading order (lod) is $\mathrm{lod}(Q) = \mathrm{ord}(x^{a'} y^{b'})$. For two polynomials $(Q, Q') \in \mathbb{F}_q[x, y]$, it holds that $Q < Q'$, if $\mathrm{lod}(Q) < \mathrm{lod}(Q')$.

3 PACD algorithm

The PACD algorithm constructs a set of interpolation test-vectors which will be ordered according to their potential for yielding the intended message. For all the test-vectors, interpolation of the common elements will be performed once and its outcome will be shared by all the test-vectors. Interpolation of the uncommon elements will then be performed progressively granting priority to decoding the test-vectors that are more likely to yield the intended message.

3.1 Construction of the interpolation test-vectors

In this paper, the binary phase shift keying (BPSK) scheme is utilised to map the RS coded bits to the modulated symbols as $c_t \mapsto s_t : 0 \mapsto \sqrt{\varepsilon},\ 1 \mapsto -\sqrt{\varepsilon}$, where $t = 0, 1, 2, \ldots, N-1$ and ε is the transmitted symbol energy. Hence, a BPSK symbol vector $\overline{\boldsymbol{s}} = (s_0, s_1, \ldots, s_{N-1})$ is transmitted. Let $\mathbb{R}$ denote the set of real numbers. After a discrete memoryless channel, the received symbol vector $\overline{\boldsymbol{R}} = (\boldsymbol{R}_0, \boldsymbol{R}_1, \ldots, \boldsymbol{R}_{n-1}) \in \mathbb{R}^N$ is obtained, where $\boldsymbol{R}_j = (r_{j\rho}, r_{j\rho+1}, \ldots, r_{(j+1)\rho-1}) \in \mathbb{R}^\rho$ and $j = 0, 1, \ldots, n-1$. With the received symbol vector $\overline{\boldsymbol{R}}$, the reliability matrix $\boldsymbol{\Pi}$ of size $q \times n$ can be further obtained. By assuming the rows of matrix $\boldsymbol{\Pi}$ are indexed by the elements of $\mathbb{F}_q$, its entries $\pi_{i,j}$ are the symbol wise *a posteriori* probabilities (APPs) defined as

$$\pi_{i,j} = \Pr[C_j = i | \boldsymbol{R}_j], \text{ for } i \in \mathbb{F}_q \text{ and } 0 \le j \le n-1. \tag{4}$$

Let $i_j^1 = \mathrm{argmax}_{i \in \mathbb{F}_q}\{\pi_{i,j}\}$ and $i_j^2 = \mathrm{argmax}_{i \in \mathbb{F}_q, i \neq i_j^1}\{\pi_{i,j}\}$ denote the row indices of the largest and the second largest entries of column j, respectively. The most and the second most likely hard-decision results of C_j are $Y_j^1 = i_j^1$ and $Y_j^2 = i_j^2$, respectively. The following metric is defined to assess the reliability of the decision on symbol C_j [13]

$$\gamma_j = \frac{\pi_{i_j^2, j}}{\pi_{i_j^1, j}} \tag{5}$$

and $\gamma_j \in (0, 1]$. As γ_j approaches one, the decision is less reliable, and vice versa. Sorting all γ_j values in ascending order yields a new symbol index sequence $j_0, j_1, \ldots, j_{k-1}, \ldots, j_{n-1}$, which indicates $\gamma_{j_0} < \gamma_{j_1} < \cdots < \gamma_{j_{k-1}} < \cdots < \gamma_{j_{n-1}}$. Let

$$\Theta = \{j_0, j_1, \ldots, j_{k-1}\} \tag{6}$$

denote the index set of the k most reliable symbols and $\Theta^{\mathrm{c}} = \{j_k, j_{k+1}, \ldots, j_{n-1}\}$. In the proposed Chase decoding algorithm, η least reliable symbols will be selected from Θ^{c}. They can be realised as either Y_j^1 or Y_j^2. Let

$$\Phi = \{j_{n-\eta}, j_{n-\eta+1}, \ldots, j_{n-1}\} \tag{7}$$

denote the index set of the selected unreliable symbols and $\Phi^{\mathrm{c}} = \{j_0, j_1, \ldots, j_{n-\eta-1}\}$. Note that $\Phi \subseteq \Theta^{\mathrm{c}}$. With the above definitions, the interpolation test-vectors can be generally written as

$$\overline{\mathcal{Y}}_v = (\mathcal{Y}_{v,0}, \mathcal{Y}_{v,1}, \ldots, \mathcal{Y}_{v,n-1}), \tag{8}$$

where

$$\mathcal{Y}_{v,j} = \begin{cases} Y_j^1, & \text{if } j \in \Phi^{\mathrm{c}}, \\ Y_j^1 \text{ or } Y_j^2, & \text{if } j \in \Phi. \end{cases} \tag{9}$$

Since there are two decisions for each of the η unreliable symbols, 2^η interpolation test-vectors will be constructed and $v = 1, 2, \ldots, 2^\eta$.

3.2 Ordering of the interpolation test-vectors

Among the 2^η interpolation test-vectors, the PACD algorithm aims to decode the ones that are more likely to yield the intended message, so that the decoding can be terminated earlier. The following reliability function is defined to assess the potential of each test-vector

$$\Omega_v = \sum_{j=0}^{n-1} \log(\pi_{\mathcal{Y}_{v,j},j}), \tag{10}$$

where the base of the logarithm is ten. A higher Ω_v value indicates test-vector $\overline{\mathcal{Y}}_v$ is more reliable and it has a higher potential for yielding the intended message. Therefore, all 2^η test-vectors will be ordered according to their reliability function Ω_v, and the one that has a larger Ω_v value will be decoded earlier. Since Ω_v can also be written as

$$\Omega_v = \sum_{j\in\Phi^c} \log(\pi_{\mathcal{Y}_{v,j},j}) + \sum_{j\in\Phi} \log(\pi_{\mathcal{Y}_{v,j},j}), \tag{11}$$

and all the reliability functions share a common part of $\sum_{j\in\Phi^c} \log(\pi_{\mathcal{Y}_{v,j},j})$, the ordering metric can be further simplified to

$$\Omega'_v = \sum_{j\in\Phi} \log(\pi_{\mathcal{Y}_{v,j},j}). \tag{12}$$

Sorting all Ω'_v values yields a new test-vector index sequence $v_1, v_2, \ldots, v_{2^\eta}$, which indicates $\Omega'_{v_1} > \Omega'_{v_2} > \cdots > \Omega'_{v_{2^\eta}}$. The PACD algorithm first decodes $\overline{\mathcal{Y}}_{v_1}$, then decodes $\overline{Y}_{v_2}$ etc. Note that the first test-vector to be decoded is the hard-decision received word $\overline{\mathcal{Y}}_{v_1} = (Y_0^1, Y_1^1, \ldots, Y_{n-1}^1)$.

3.3 Common element interpolation

For a test-vector $\overline{\mathcal{Y}}_v$, interpolation constructs a minimal polynomial $Q(x, y)$ that satisfies $Q(x_j, \mathcal{Y}_{v,j}) = 0$ for all j. Since all the interpolation test-vectors share the common symbols Y_j^1 for $j\in\Phi^c$, interpolation will first be performed for those common elements. Its outcome will be shared by all the test-vectors. In this paper, we will discuss the test-vectors $\overline{\mathcal{Y}}_v$ of the form $\overline{\mathcal{Y}}_v = (\mathcal{Y}_{v,j_0}, \mathcal{Y}_{v,j_1}, \ldots, \mathcal{Y}_{v,j_{n-1}})$. The interpolation order for the n points is $(x_{j_0}, \mathcal{Y}_{v,j_0}) \to (x_{j_1}, \mathcal{Y}_{v,j_1}) \to \cdots \to (x_{j_{n-1}}, \mathcal{Y}_{v,j_{n-1}})$.

The common element interpolation is assisted by the re-encoding transform. The re-encoding polynomial is defined as [11]

$$\Psi(x) = \sum_{j\in\Theta} Y_j^1 \psi_j(x), \tag{13}$$

where

$$\psi_j(x) = \prod_{(j,\delta)\in\Theta, j\neq\delta} \frac{x - x_\delta}{x_j - x_\delta}. \tag{14}$$

The re-encoding polynomial implies $\Psi(x_j) = Y_j^1$ for $j\in\Theta$. Therefore, given a test-vector $\overline{\mathcal{Y}}_v$, by performing the re-encoding transform

$$\mathcal{Y}'_{v,j} = \mathcal{Y}_{v,j} - \Psi(x_j) \tag{15}$$

for all of its entries, it can be transformed into

$$\overline{\mathcal{Y}}_{v'} = (0, \ldots, 0, \mathcal{Y}'_{v,j_k}, \ldots, \mathcal{Y}'_{v,j_{n-1}}). \tag{16}$$

Interpolation for points $(x_j, 0)$ where $j\in\Theta$ can be determined by

$$V(x) = \prod_{j\in\Theta} (x - x_j). \tag{17}$$

Interpolation can then begin by initialising the following polynomial set

$$\mathcal{G}^* = \{g_1^*(x, y), g_2^*(x, y)\} = \{V(x), y\}. \tag{18}$$

Since $\mathcal{G}* = \{V(x)\cdot 1, V(x)\cdot(y/V(x))\}$, the initialised polynomial set can be further simplified to

$$\mathcal{G} = \{g_1(x, y), g_2(x, y)\} = \{1, y\}, \tag{19}$$

and the remaining interpolation points $(x_j, \mathcal{Y}'_{v,j})$ for $j\in\Theta^c$ are transformed into

$$(x_j, \mathcal{Y}''_{\mathrm{v,j}}) = \left(x_j, \frac{\mathcal{Y}'_{v,j}}{V(x_j)}\right). \tag{20}$$

After the re-encoding transform, polynomial set $\mathcal{G}$ will further interpolate points $(x_j, \mathcal{Y}''_{v,j})$ for $j\in\Theta^c$.

Since $\eta \leqslant n - k$, the test-vectors can share more than k common symbols. Let us define

$$A_c = \Theta^c \cap \Phi^c = \{j_k, j_{k+1}, \ldots, j_{n-\eta-1}\}, \tag{21}$$

$$A_u = \Theta^c \cap \Phi = \{j_{n-\eta}, j_{n-\eta+1}, \ldots, j_{n-1}\}. \tag{22}$$

In the common element interpolation, polynomials of $\mathcal{G}$ will further interpolate points $(x_j, \mathcal{Y}''_{v,j})$ for $j\in A_c$. For $(x_j, \mathcal{Y}''_{v,j})$, the polynomials' interpolation property can be judged by $g_1(x_j, \mathcal{Y}''_{v,j})$ and $g_2(x_j, \mathcal{Y}''_{v,j})$ [6]. Hence, let

$$f(x, y) = \min\{g_i(x, y) \in \mathcal{G} | g_i(x_j, \mathcal{Y}''_{v,j}) \neq 0 \text{ and } i = 1, 2\}, \tag{23}$$

and

$$g(x, y) = \mathcal{G}\backslash\{f(x, y)\}. \tag{24}$$

By performing the following bilinear modifications

$$g'(x, y) = g(x, y) - \frac{g(x_j, \mathcal{Y}''_{v,j})}{f(x_j, \mathcal{Y}''_{v,j})}\cdot f(x, y), \tag{25}$$

$$f'(x, y) = (x - x_j)\cdot f(x, y), \tag{26}$$

the updated polynomials will interpolate the point. Repeating the above process for all the points that are indexed by A_c, an updated polynomial set $\mathcal{G}$ is obtained and

$$\tilde{\mathcal{G}} = \{g_i(x, y) | g_i(x_j, \mathcal{Y}''_{v,j}) = 0, \ \forall j \in A_c \text{ and } i = 1, 2\}. \tag{27}$$

It will be utilised by the following progressive uncommon element interpolation.

3.4 Progressive uncommon element interpolation

The progressive uncommon element interpolation is first performed for test-vector $\overline{\mathcal{Y}}'_{v_1}$ by interpolating points $(x_j, \mathcal{Y}''_{v_1,j})$ for $j\in A_u$. If the intended message can be found by factorising the interpolation outcome, the decoding will be terminated. Otherwise, it will be further performed for test-vectors $\overline{\mathcal{Y}}'_{v_2}$, $\overline{\mathcal{Y}}'_{v_3}$ etc. For a test-vector

$\mathcal{Y}'_v$, we define

$$\mathcal{G}_v^{(\tau)} = \{g_i(x, y) | g_i(x_j, \mathcal{Y}''_{v,j}) = 0, j = j_k, j_{k+1}, \dots, j_{n-\eta+\tau-1} \text{ and } i = 1, 2\}. \tag{28}$$

The polynomial set that has further interpolated τ points $(x_j, \mathcal{Y}''_{v,j})$ are indexed by A_u and $0 \leqslant \tau \leqslant \eta$. When $\tau = 0$, $\mathcal{G}_v^{(0)} = \widetilde{\mathcal{G}}$ because all the test-vectors inherit the common element interpolation result. With this notation, the PACD algorithm generates the polynomial sets in the following order:

$$\mathcal{G}_{v_1}^{(1)} \to \mathcal{G}_{v_1}^{(2)} \to \cdots \to \mathcal{G}_{v_1}^{(\eta)}, \quad \mathcal{G}_{v_2}^{(1)} \to \mathcal{G}_{v_2}^{(2)} \to \cdots \to \mathcal{G}_{v_2}^{(\eta)}, \ \dots, \ \mathcal{G}_{v_{2^\eta}}^{(1)} \to \mathcal{G}_{v_{2^\eta}}^{(2)} \to \cdots \to \mathcal{G}_{v_{2^\eta}}^{(\eta)}.$$

In general, those polynomial sets can be denoted as $\mathcal{G}_{v_\varpi}^{(\tau)}$, where $\varpi = 1, 2, \dots, 2^\eta$. If all 2^η test-vectors are interpolated, $\eta \cdot 2^\eta$ polynomial sets will be generated. The following proposition shows that they are not unique and the PACD algorithm reduces the decoding computation utilising this property.

Proposition 1: For two test-vectors $\mathcal{Y}'_{v_\varpi}$ and $\mathcal{Y}'_{v_{\varpi'}}$, if $\mathcal{Y}''_{v_\varpi, j} = \mathcal{Y}''_{v_{\varpi'}, j}$ for $j = j_{n-\eta}, j_{n-\eta+1}, \dots, j_{n-\eta+\tau-1}$, then $\mathcal{G}_{v_\varpi}^{(1)} = \mathcal{G}_{v_{\varpi'}}^{(1)}$, $\mathcal{G}_{v_\varpi}^{(2)} = \mathcal{G}_{v_{\varpi'}}^{(2)}$, $\dots$, $\mathcal{G}_{v_\varpi}^{(\tau)} = \mathcal{G}_{v_{\varpi'}}^{(\tau)}$.

Without referring to the test-vector ordering, polynomial set $\mathcal{G}_v^{(\tau)}$ is obtained by interpolating the polynomials of $\mathcal{G}_v^{(\tau-1)}$ for either of the following two points:

$$P_\tau^1 = \left(x_{j_{n-\eta+\tau-1}}, \frac{Y^1_{j_{n-\eta+\tau-1}} - \Psi(x_{j_{n-\eta+\tau-1}})}{V(x_{j_{n-\eta+\tau-1}})}\right),$$

$$P_\tau^2 = \left(x_{j_{n-\eta+\tau-1}}, \frac{Y^2_{j_{n-\eta+\tau-1}} - \Psi(x_{j_{n-\eta+\tau-1}})}{V(x_{j_{n-\eta+\tau-1}})}\right).$$

If all the test-vectors are interpolated, the evolution of the polynomial sets $\mathcal{G}_v^{(\tau)}$ with τ growing from zero to η can be illustrated as a binary tree that is shown in Fig. 1. At layer τ, there are 2^τ distinct polynomial sets $\mathcal{G}_v^{(\tau)}$, each of which is shared by $2^{\eta-\tau}$ test-vectors. A complete path from the *root* $\mathcal{G}_v^{(0)}$ to a *leaf* $\mathcal{G}_v^{(\eta)}$ indicates the uncommon element interpolation for a particular test-vector $\mathcal{Y}'_v$. The LCC algorithm [13] results in the binary tree growing in a *layer-by-layer* manner. This implies that at layer τ, polynomial sets $\mathcal{G}_v^{(\tau)}$ will be fully determined by interpolating the polynomial sets $\mathcal{G}_v^{(\tau-1)}$. Finally, 2^η polynomial sets $\mathcal{G}_v^{(\eta)}$ will be generated. They are the uncommon element interpolation outcomes of the 2^η test-vectors.

In contrast, the PACD algorithm grows the binary tree in a *depth-first-search* manner. It first generates a complete path from $\mathcal{G}_{v_1}^{(0)}$ to $\mathcal{G}_{v_1}^{(\eta)}$. If the intended message can be found by factorising the minimal polynomial of $\mathcal{G}_{v_1}^{(\eta)}$, the decoding will be terminated. Otherwise, it generates another path from $\mathcal{G}_{v_2}^{(0)}$ to $\mathcal{G}_{v_2}^{(\eta)}$ etc. Note that the ML criterion that is stated as Lemma 1 of [19] will be utilised to validate the factorisation output. However, based on *Proposition 1*, it can be realised that the interpolation of a later decoded test-vector $\mathcal{Y}'_{v_\varpi}$ with $\varpi > 1$ does not necessarily need to start from $\mathcal{G}_{v_\varpi}^{(0)}$. It can utilise the intermediate interpolation information that is generated and stored during decoding of the previous $\varpi - 1$ test-vectors. The intermediate interpolation information is defined as

$$\mathcal{M}_{\varpi-1} = \{\mathcal{G}_{v_{\varpi'}}^{(\tau)}, \ 0 \le \tau < \eta \ \text{ and } \ 1 \le \varpi' \le \varpi - 1\}. \tag{29}$$

Note that $\mathcal{G}_{v_{\varpi'}}^{(\eta)}$ does not need to be stored since it will not be utilised by a later decoded test-vector.

Without loss of generality, we now describe the uncommon element interpolation of a test-vector $\mathcal{Y}'_{v_\varpi}$, where $\varpi = 1, 2, \dots, 2^\eta$. If $\varpi = 1$, it is the first test-vector to be decoded. There is no

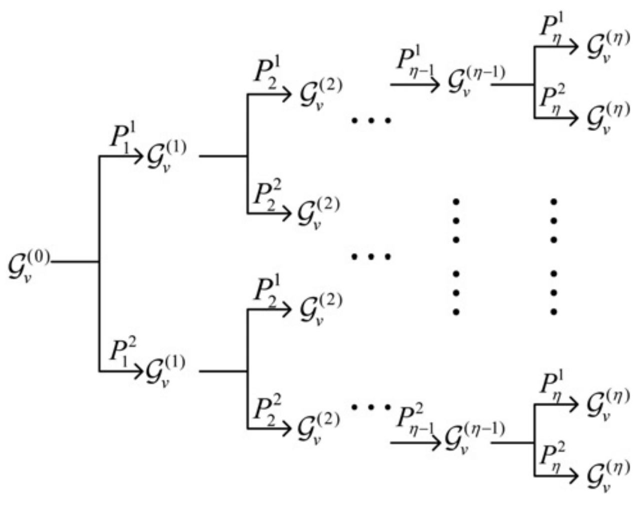

Fig. 1 *Binary tree representation of the uncommon element interpolation*

stored interpolation information and $\mathcal{M}_0 = \emptyset$. Its uncommon element interpolation will have to start from the *root* of the binary tree, i.e. $\mathcal{G}_{v_1}^{(0)}$. It will then interpolate points $(x_j, \mathcal{Y}''_{v_1,j})$ that are indexed by A_u, yielding polynomial set $\mathcal{G}_{v_1}^{(\eta)}$. Consequently, the stored information is updated to $\mathcal{M}_1$. Otherwise, if $\varpi > 1$, we will first assess its similarity with the decoded test-vectors. In the binary tree, let $\Xi_\varpi(\varpi')$ denote the number of layers shared by $\mathcal{Y}'_{v_\varpi}$ and $\mathcal{Y}'_{v_{\varpi'}}$ $(\varpi' < \varpi)$ as

$$\Xi_\varpi(\varpi') = \max\{\tau | \mathcal{Y}''_{v_\varpi, j_{n-\eta+\tau'-1}} = \mathcal{Y}''_{v_{\varpi'}, j_{n-\eta+\tau'-1}}, \text{ for } 0 \le \tau' \le \tau\}. \tag{30}$$

The decoded test-vectors $\mathcal{Y}'_{v_{\varpi'}}$ that share the maximal number of layers with $\mathcal{Y}'_{v_\varpi}$ should be identified and denoted as

$$\{\mathcal{Y}'_{v_{\varpi^*}} : \varpi^* = \underset{\varpi' < \varpi}{\operatorname{argmax}} \{\Xi_\varpi(\varpi')\}\}. \tag{31}$$

Note that more than one decoded test-vector can be identified, but only one of them will be selected and denoted as $\mathcal{Y}'_{v_{\varpi^*}}$. Recalling *Proposition 1*, we know that $\mathcal{G}_{v_\varpi}^{(\tau)} = \mathcal{G}_{v_{\varpi^*}}^{(\tau)}$ for $\tau = 0, 1, \dots, \Xi_\varpi(\varpi^*)$, while $\mathcal{G}_{v_{\varpi^*}}^{(0)}$, $\mathcal{G}_{v_{\varpi^*}}^{(1)}$, $\dots$, and $\mathcal{G}_{v_{\varpi^*}}^{(\Xi_\varpi(\varpi^*))}$ are the stored information of $\mathcal{M}_{\varpi-1}$. Therefore, to generate $\mathcal{G}_{v_\varpi}^{(\eta)}$, we will first initialise

$$\mathcal{G}_{v_\varpi}^{(\Xi_\varpi(\varpi^*))} = \mathcal{G}_{v_{\varpi^*}}^{(\Xi_\varpi(\varpi^*))}. \tag{32}$$

Polynomial set $\mathcal{G}_{v_\varpi}^{(\Xi_\varpi(\varpi^*))}$ will then interpolate points $(x_j, \mathcal{Y}''_{v_\varpi,j})$ for $j = j_{n-\eta+\Xi_\varpi(\varpi^*)}, \dots, j_{n-1}$, yielding polynomial set $\mathcal{G}_{v_\varpi}^{(\eta)}$. The stored information is further updated into $\mathcal{M}_\varpi$.

After completing the uncommon element interpolation for a test-vector $\mathcal{Y}'_{v_\varpi}$, the minimal polynomial of $\mathcal{G}_{v_\varpi}^{(\eta)}$, i.e.

$$\widetilde{Q}_{v_\varpi}(x, y) = \min\{g_i(x, y) \in \mathcal{G}_{v_\varpi}^{(\eta)}, \ i = 1, 2\} \tag{33}$$

is chosen. Since

$$\widetilde{Q}_{v_\varpi}(x, y) = \tilde{q}_{v_\varpi,0}(x) + y \cdot \tilde{q}_{v_\varpi,1}(x), \tag{34}$$

and polynomial $V(x)$ has been extracted from set $\mathcal{G}^*$ at the beginning of the common element interpolation, the interpolated polynomial $Q_{v_\varpi}(x, y)$ should be reconstructed by

$$Q_{v_\varpi}(x, y) = \tilde{q}_{v_\varpi,0}(x) \cdot V(x) + y \cdot \tilde{q}_{v_\varpi,1}(x). \tag{35}$$

```
Algorithm 1
Input: $\eta$;
Output: A list of message candidates $\hat{M}(x)$ or $\emptyset$;
Initialisation: $\varpi = 1$ and $\mathcal{M}_0 = \emptyset$;
 1: Construct $2^\eta$ test-vectors $\overline{\mathcal{Y}}_v$;
 2: Order all the $2^\eta$ test-vectors according to their $\Omega'_v$ values;
 3: Perform the re-encoding transform as in (15);
 4: Initialise $\mathcal{G}$ as in (19) and interpolate points $(x_j, \mathcal{Y}''_{v,j})$ where $j \in \mathrm{A}_c$ as in (23)-(26);
 5: for the test-vector $\overline{\mathcal{Y}}'_{v_\varpi}$ do
 6:     if $\varpi = 1$ then
 7:         Let $\mathcal{G}_{v_1}^{(0)} = \widetilde{\mathcal{G}}$;
 8:         Interpolate points $(x_j, \mathcal{Y}''_{v_1,j})$ where $j \in \mathrm{A}_u$ as in (23)-(26);
 9:     else
10:         Determine $\Xi_\varpi(\varpi')$ as in (30);
11:         Identify a decoded test-vector $\overline{\mathcal{Y}}'_{v_{\varpi^*}}$ as in (31);
12:         Let $\mathcal{G}_{v_\varpi}^{(\Xi_\varpi(\varpi^*))} = \mathcal{G}_{v_{\varpi^*}}^{(\Xi_\varpi(\varpi^*))}$;
13:         Interpolate points $(x_j, \mathcal{Y}''_{v_\varpi,j})$ for $j = j_{n-\eta+\Xi_\varpi(\varpi^*)}, \ldots, j_{n-1}$ as in (23)-(26);
14:     end if
15:     Find the minimal polynomial $\widetilde{Q}_{v_\varpi}(x, y)$ of (33);
16:     Restore polynomial $Q_{v_\varpi}(x, y)$ as in (35);
17:     Factorize $Q_{v_\varpi}(x, y)$ to obtain $M'(x)$;
18:     Estimate $\widehat{M}(x)$ as in (36);
19:     Perform the re-encoding of $\widehat{M}(x)$;
20:     if the codeword satisfies the ML criterion then
21:         Output $\widehat{M}(x)$ and terminate the decoding;
22:     else
23:         Update $\varpi = \varpi + 1$ and go to step 5;
24:     end if
25: end for
```

Fig. 2 *The PACD algorithm for RS codes*

It satisfies $Q_{v_\varpi}(x_j, \mathcal{Y}'_{v_\varpi,j}) = 0$ for all j. A message polynomial $M'(x)$ that is in the form of (1) can be obtained by factorising $Q_{v_\varpi}(x, y)$ [22]. An estimation of the intended message polynomial is further generated by

$$\widehat{M}(x) = M'(x) + \Psi(x). \tag{36}$$

If the re-encoding of $\widehat{M}(x)$ yields an ML codeword, the decoding will be terminated. Otherwise, the next test-vector $\overline{\mathcal{Y}}_{v_{\varpi+1}}$ will be decoded based on the memorised information $\mathcal{M}_\varpi$. If none of the 2^η message candidates yields an ML codeword, the PACD algorithm results in a full growth of the binary tree. Among all the message candidates, the one that yields the most likely codeword will be selected. Summarising the above descriptions, the PACD algorithm is presented as in Algorithm 1 (see Fig. 2).

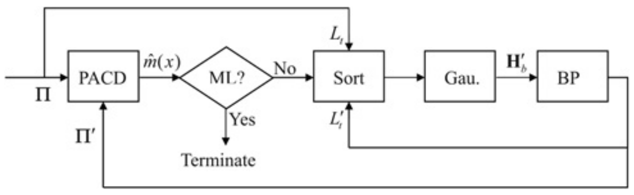

Fig. 3 *Block diagram of the E-PACD algorithm*

4 E-PACD algorithm

For the PACD algorithm, increasing η will lead to a stronger error-correction capability, since there are more test-vectors to be decoded. However, this also increases the decoding complexity exponentially. To improve the PACD algorithm's decoding performance while maintaining a moderate decoding complexity, the E-PACD algorithm is introduced by coupling the PACD algorithm with the ABP decoding. The ABP decoding is functioning with an adaptive binary parity-check matrix of an RS code. The adaptive matrix is sparser than the original parity-check matrix $\boldsymbol{H}_b$, making it more suitable to be applied for belief propagation. References [20, 21] have shown that ABP decoding can improve the reliability of the received information, after which an algebraic RS decoding algorithm can be used to achieve a superior decoding performance. Therefore, this paper integrates the PACD and the ABP decoder, where the ABP decoder can generate more test-vectors for the PACD algorithm. More importantly, the E-PACD algorithm's complexity can be significantly reduced by eliminating the decoded test-vectors and utilising the information that is generated during the previous PACD attempts.

The block diagram of the E-PACD algorithm is presented in Fig. 3. The PACD algorithm is performed based on the original reliability matrix $\mathbf{\Pi}$. If the intended message cannot be found, the ABP decoder [20, 21] is applied again to generate an enhanced reliability matrix $\mathbf{\Pi}'$ which results in new test-vectors to be decoded. The ABP decoder consists of bit reliability sorting, Gaussian elimination and BP decoding, forming an iterative process. Each ABP iteration yields an enhanced matrix $\mathbf{\Pi}'$ for the next progressive Chase decoding in finding the intended message. The ABP decoding can be briefly described as follows.

With the received vector $\overline{\boldsymbol{R}}$, the bit wise APP values $\Pr[c_t = 0|r_t]$ and $\Pr[c_t = 1|r_t]$ can be obtained. The log-likelihood ratio (LLR) of c_t is defined as

$$L_t = \ln \frac{\Pr[c_t = 0|r_t]}{\Pr[c_t = 1|r_t]}, \tag{37}$$

where $t = 0,\ 1,\ \dots,\ N-1$. A larger $|L_t|$ implies bit c_t is more reliable, and vice versa. Therefore, sorting all N LLR magnitudes in an ascending order yields a refreshed bit index sequence $t_0,\ t_1,\ \dots,\ t_{N-K-1},\ \dots,\ t_{N-1}$, which indicates $|L_{t_0}| < |L_{t_1}| < \cdots < |L_{t_{N-K-1}}| < \cdots < |L_{t_{N-1}}|$. Gaussian elimination will then be performed on the binary parity-check matrix $\boldsymbol{H}_\mathrm{b}$, reducing the columns that correspond to the $N-K$ unreliable bits to weight-1 columns. It results in an adapted matrix $\boldsymbol{H}'_\mathrm{b}$ that is sparser. The iterative BP decoding will be performed based on $\boldsymbol{H}'_\mathrm{b}$. By defining $\boldsymbol{\Lambda}(t) \triangleq \{\lambda | h_{\lambda t} = 1,\ \forall h_{\lambda t} \in \boldsymbol{H}'_\mathrm{b}\}$ and $\boldsymbol{T}(\lambda) \triangleq \{t | h_{\lambda t} = 1,\ \forall h_{\lambda t} \in \boldsymbol{H}'_\mathrm{b}\}$, the BP decoding updates the bit wise LLRs by

$$L'_t = L_t + \vartheta \cdot \sum_{\lambda \in \boldsymbol{\Lambda}(t)} 2\tanh^{-1}\left(\prod_{t' \in \boldsymbol{T}(\lambda)\backslash t} \tanh\left(\frac{L_{t'}}{2}\right)\right), \tag{38}$$

where $\vartheta \in (0,\ 1]$ is the damping factor [20]. The updated LLRs will be fed back for another sorting process, triggering the next round of ABP decoding. In the meantime, each updated LLR will be converted back to a pair of bit wise APP values by

$$\Pr[c_t = 0|r_t] = \frac{1}{1 + \mathrm{e}^{-L'_t}}, \quad \Pr[c_t = 1|r_t] = \frac{1}{1 + \mathrm{e}^{L'_t}}, \tag{39}$$

with which an enhanced reliability matrix $\mathbf{\Pi}'$ can be formed. Given a defined η value, there will be 2^η test-vectors for the next round of PACD.

As shown in Fig. 3, if the intended message cannot be found from the original PACD attempt, the ABP decoder is deployed. After each ABP iteration, an enhanced matrix $\mathbf{\Pi}'$ is formed. By storing the image of the decoded test-vectors $\overline{\mathcal{Y}}_v$, the newly generated test-vectors are compared with the decoded vectors. This implies that without performing the re-encoding transform, if the n interpolation points $(x_j,\ \mathcal{Y}_{v,j})$ of a new test-vector are identical to those from a decoded test-vector, the new vector is redundant and is eliminated from the current PACD attempt. To further reduce the redundant computations during the multiple PACD attempts, the following information of each PACD attempt will be stored. It includes the symbol index sets Θ, A_c and A_u, polynomials $\Psi(x)$ and $V(x)$ and polynomial set $\tilde{\mathcal{G}}$. Fig. 4 shows that for a test-vector $\overline{\mathcal{Y}}_{v'}$, polynomials $\Psi(x)$ and $V(x)$ are defined by set Θ, while polynomial set $\tilde{\mathcal{G}}$ is defined by both Θ and A_c. Note that for $j \in \Theta^\mathrm{c}$, symbols $\mathcal{Y}''_{v,j}$ are transformed from symbols $\mathcal{Y}'_{v,j}$ by $\mathcal{Y}''_{v,j} = \dfrac{\mathcal{Y}'_{v,j}}{V(x_j)}$.

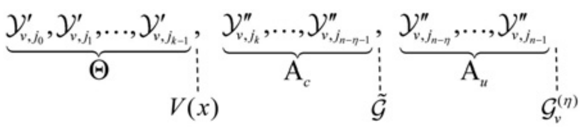
$\underbrace{\mathcal{Y}'_{v,j_0}, \mathcal{Y}'_{v,j_1}, \dots, \mathcal{Y}'_{v,j_{k-1}}}_{\Theta}$, $\underbrace{\mathcal{Y}''_{v,j_k}, \dots, \mathcal{Y}''_{v,j_{n-\eta-1}}}_{A_c}$, $\underbrace{\mathcal{Y}''_{v,j_{n-\eta}}, \dots, \mathcal{Y}''_{v,j_{n-1}}}_{A_u}$
$V(x)$
$\tilde{\mathcal{G}}$
$\mathcal{G}_v^{(\eta)}$

Fig. 4 *Test-vector and the interpolation information*

We now use κ to denote the index of each PACD attempt. The above-mentioned information that is associated with each PACD attempt is further denoted as Θ_κ, $A_{\mathrm{c},\kappa}$, $A_{u,\kappa}$, $\Psi_\kappa(x)$, $V_\kappa(x)$ and $\tilde{\mathcal{G}}_\kappa$. When $\kappa = 1$, it implies that this is the original progressive Chase decoder without deploying the ABP assistance. When $\kappa > 1$, we will first compare the image of the current test-vectors and the decoded ones, eliminating the decoded test-vectors. For the remaining test-vectors, the re-encoding transform will be performed and the index sets Θ_κ, $A_{\mathrm{c},\kappa}$ and $A_{u,\kappa}$ will be defined. Operation of the current PACD attempt can be categorised into the following three scenarios:

Scenario I: If $\exists\kappa'$ and $1 \leqslant \kappa' < \kappa$, $\Theta_{\kappa'} = \Theta_\kappa$ and the symbols $\mathcal{Y}''_{v,j}$ with $j \in A_{\mathrm{c},\kappa}$ are identical to those defined by $A_{\mathrm{c},\kappa'}$, both polynomials $\Psi_{\kappa'}(x)$ and $V_{\kappa'}(x)$, and the polynomial set $\tilde{\mathcal{G}}_{\kappa'}$ can be utilised. For the current PACD attempt, its decoding can start with $\tilde{\mathcal{G}}_\kappa = \tilde{\mathcal{G}}_{\kappa'}$ and further interpolate points $(x_j,\ \mathcal{Y''}_{v,j})$ for $j \in A_{u,\kappa}$.
Scenario II: If *Scenario I* does not occur, but $\exists\kappa'$ and $1 \leqslant \kappa' < \kappa$, $\Theta_{\kappa'} = \Theta_\kappa$, the polynomials $\Psi_{\kappa'}(x)$ and $V_{\kappa'}(x)$ can be utilised. The current PACD attempt can start its decoding with $\mathcal{G} = \{1,\ y\}$ and further interpolate points $(x_j,\ \mathcal{Y}''_{v,j})$ for $j \in A_{\mathrm{c},\kappa} \cup A_{u,\kappa}$.
Scenario III: If $\forall\kappa'$ and $1 \leqslant \kappa' < \kappa$, $\Theta_{\kappa'} \neq \Theta_\kappa$, none of the memorised information can be utilised. The current PACD will have to start by performing the re-encoding transform with defining $\Phi_\kappa(x)$ and $V_\kappa(x)$. It then initialises polynomial set $\mathcal{G} = \{1,\ y\}$ and further interpolates points $(x_j,\ \mathcal{Y}''_{v,j})$ for $j \in A_{\mathrm{c},\kappa} \cup A_{u,\kappa}$.

Note that in *Scenarios I* and *II*, the interpolated polynomial will be reconstructed by utilising $V_{\kappa'}(x)$ as in (35). The intended message polynomial can be further recovered by utilising $\Psi_{\kappa'}(x)$ as in (36). While in *Scenario III*, the above-mentioned operations will have to utilise $V_\kappa(x)$ and $\Psi_\kappa(x)$ of the current PACD attempt.

5 Performance analysis

This section analyses the error-correction performance of the PACD and the E-PACD algorithms. Figs. 5–7 show their performance in decoding the (15, 11), the (31, 27) and the (255, 239) RS codes over the additive white Gaussian noise (AWGN) channel, respectively. The damping factor that optimises the E-PACD performance for the code is also indicated in these figures. In the simulations, the E-PACD algorithm functions with three ABP iterations and there is one BP iteration for each adapted matrix $\boldsymbol{H}'_\mathrm{b}$. It is compared with the GS algorithm with an interpolation multiplicity of one. For the mentioned RS codes, GS decoding with multiplicity of one reaches the hard-decision list decoding bound of $n - \lfloor\sqrt{n(k-1)}\rfloor - 1$. It is also compared with the KV algorithm with a designed factorisation OLS of three, which achieves its optimal performance [The optimal KV decoding performance is obtained by (28) of [7]]. The ML decoding bounds are also given as benchmarks to assess the error-correction potential of the E-PACD algorithm. In the following discussion, coding gains are evaluated at a frame error rate (FER) of 10^{-4}.

Our simulation results show that by increasing η, performance of the PACD algorithm can be improved, since more test-vectors are decoded. In particular, by increasing η from one to four, coding gains of 1.2 and 1 dB can be further achieved for the (15, 11) and the (31, 27) RS codes, respectively. However, for the (255, 239) RS code, less significant coding gains are obtained by increasing η. This is because the (255, 239) RS code has a much larger codebook cardinality. For an $(n,\ k)$ RS code defined in $\mathbb{F}_q$, its codebook cardinality is q^k. The PACD algorithm decodes 2^η

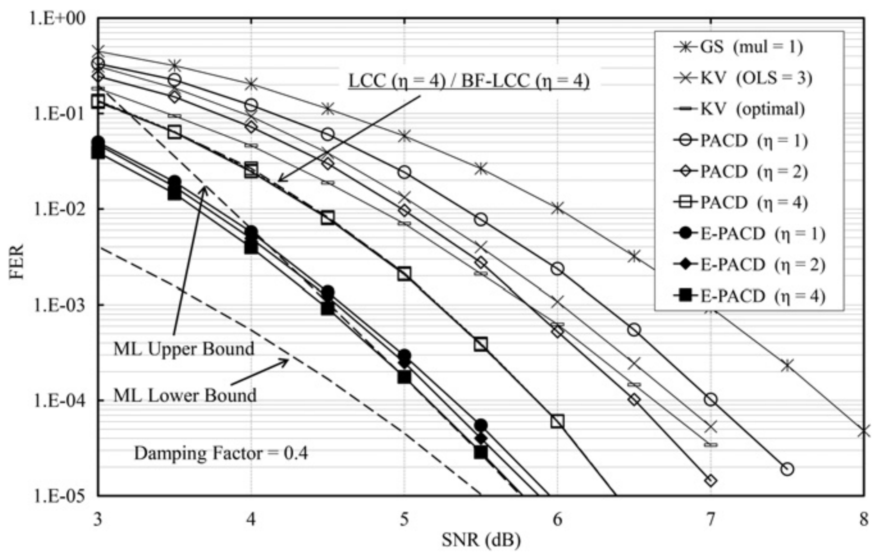

Fig. 5 *Performance of the (15, 11) RS code over the AWGN channel*

test-vectors, attempting to recover the transmitted codeword from one of them. With the same η value, a larger codebook cardinality can lead to a lower probability of recovering the transmitted codeword. Consequently, Chase decoding plays a less significant role in enhancing the error-correction performance. For the (15, 11) and (31, 27) RS codes, the PACD algorithm with $\eta = 2$ outperforms the optimal KV performance. While for the (255, 239) RS code, the PACD algorithm can only outperform the optimal KV performance when $\eta = 7$. It should be pointed out that with the same η value and the same decoded message selection criterion, the PACD algorithm has the same performance as the LCC algorithm [13] and the BF-LCC algorithm [14]. This is evidenced by Fig. 5, which shows the LCC and the BF-LCC performances with $\eta = 4$.

The E-PACD algorithm further improves the error-correction performance. By performing ABP decoding, the received information is enhanced resulting in a stronger Chase decoding capability. For both the (15, 11) and (31, 27) RS codes, the E-PACD when $\eta = 1$ outperforms the PACD when $\eta = 4$. Given there are at most three ABP iterations, the E-PACD algorithm when $\eta = 1$ will decode at most eight test-vectors while the PACD algorithm when $\eta = 4$ would have to decode up to 16 test-vectors. It indicates that with extra ABP decoding, the E-PACD algorithm can achieve a better performance with less finite field arithmetic operations. A similar performance improvement of the E-PACD algorithm can also be observed for the (255, 239) RS code. However, for the E-PACD algorithm, increasing η does not yield the same coding gain as for the PACD algorithm. This is because the E-PACD performance gains mainly come from the ABP decoding that enhances the reliability of the received information, and the role of parameter η plays a less significant role. Moveover, Figs. 5 and 6 show that for the (15, 11) and the (31, 27) RS codes, the E-PACD performances are close to the ML decoding bounds [23]. It is important to point out that the performance gains offered by the PACD and the E-PACD algorithms over the benchmark decoding approaches are realised in a more efficient decoding manner. The following section further analyses the computational complexity of the PACD algorithm and also sheds light on the complexity of the E-PACD algorithm.

6 Complexity analysis

Since the PACD algorithm gives priority to decoding the higher potential test-vectors and terminates once the intended message is found, it reduces the complexity that would otherwise be spent on decoding all 2^{η} test-vectors. Therefore, with less corrupted received information, the intended message can be decoded earlier

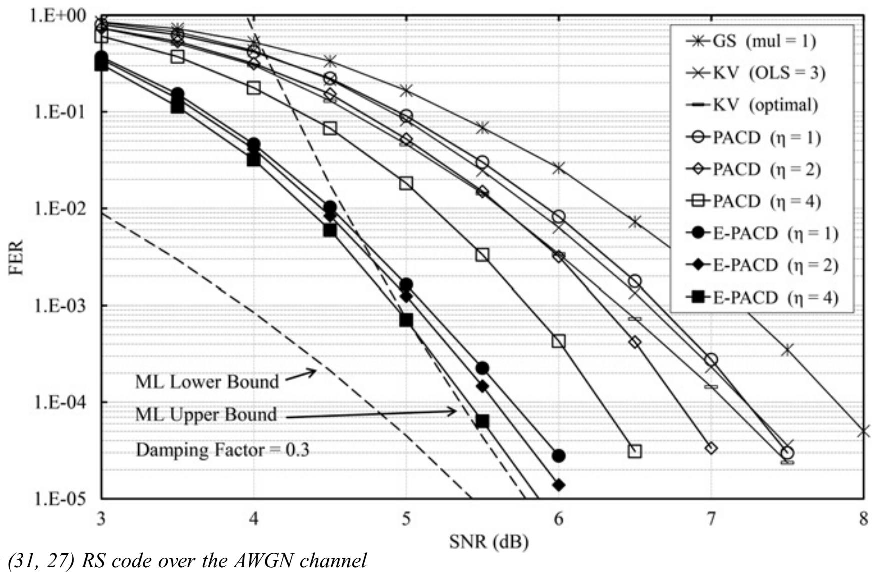

Fig. 6 *Performance of the (31, 27) RS code over the AWGN channel*

with a lower decoding complexity, which is measured as the number of finite field arithmetic operations to decode an RS codeword.

For the PACD algorithm, the complexity of each decoding event varies as it can terminate after decoding any of the 2^η test-vectors. We define the worst case decoding event as all 2^η test-vectors being decoded. In contrast, the best case decoding event would be only one test-vector that has been decoded. The complexity of the two decoding events will be analysed and they should envelop the actual decoding complexity. In the proposal, interpolation consists of the re-encoding, the common element interpolation and the uncommon element interpolation, for which we use $\mathcal{C}_{\text{int}}^{(1)}$, $\mathcal{C}_{\text{int}}^{(2)}$ and $\mathcal{C}_{\text{int}}^{(3)}$ to denote their complexity, respectively. The interpolation complexity is

$$\mathcal{C}_{\text{int}} = \mathcal{C}_{\text{int}}^{(1)} + \mathcal{C}_{\text{int}}^{(2)} + \mathcal{C}_{\text{int}}^{(3)}. \tag{40}$$

Further considering the complexity of factorisation $\mathcal{C}_{\text{fac}}$ and the ML codeword validation $\mathcal{C}_{\text{val}}$, the PACD algorithm's complexity in decoding a test-vector can be formulated as

$$\mathcal{C}_{\text{PACD}} = \mathcal{C}_{\text{int}} + \mathcal{C}_{\text{fac}} + \mathcal{C}_{\text{val}}. \tag{41}$$

For the re-encoding, calculating the numerator and denominator of the Lagrange basis polynomial $\psi_j(x)$ of (14) requires $\sum_{i=2}^{k-1} 2i = (k+1)(k-2)$ and $2(k-1)$ finite field operations, respectively. Multiplying $\psi_j(x)$ by y_j^1 further requires k finite field operations. The re-encoding polynomial $\Psi(x)$ of (13) is a sum of k Lagrange basis polynomials. Hence, forming $\Psi(x)$ requires $k(2(k-1)+(k+1)(k-2)+k)+k = k(k+3)(k-1)$ finite field operations. Furthermore, calculating polynomial $V(x)$ of (17) requires $\sum_{i=2}^{k} 2i = (k+2)(k-1)$ finite field operations. Substituting x_j with $j \in \overline{\Theta}^c$ into $\Psi(x)$ and $V(x)$ requires $3(k-1)(n-k)$ and $3k(n-k)$ finite field operations, respectively. Therefore, the re-encoding complexity is

$$\begin{aligned}\mathcal{C}_{\text{int}}^{(1)} &= k(k+3)(k-1) + (k+2)(k-1) + (6k-3)(n-k) \\ &\simeq (k+1)^3 + 6k(n-k).\end{aligned} \tag{42}$$

The interpolation complexity is proportional to the size of the polynomial, e.g. given $g_0(x, y)$ and $g_1(x, y)$ both of which have i coefficients, calculating $g_0(x_j, \mathcal{Y}''_{v,j})$ and $g_1(x_j, \mathcal{Y}''_{v,j})$ and updating the two polynomials as in (25) and (26) require at least $6i$ and $4i$ finite field operations, respectively. Considering size of polynomials grows as they interpolate the points, and there are $n-k-\eta$ points for the common element interpolation, its complexity is

$$\mathcal{C}_{\text{int}}^{(2)} = \sum_{i=1}^{n-k-\eta} 10i = 5(n-k-\eta)(n-k-\eta+1). \tag{43}$$

Note that the re-encoding and the common element interpolation will be performed once and are shared by all the test-vectors.

In the worst-case decoding event, the uncommon element interpolation results in a full expansion of the binary tree (see Fig. 1). There are 2^τ polynomial groups at layer τ. Each polynomial in the layer has approximately $n-k-\eta+\tau+1$ coefficients. Interpolation of a point requires $10(n-k-\eta+\tau+1)$ finite field operations. With $\tau < \eta$, each polynomial group will interpolate two points P_τ^1 and P_τ^2. Therefore, the uncommon element interpolation complexity is

$$\begin{aligned}\mathcal{C}_{\text{int}}^{(3)} &= \sum_{\tau=0}^{\eta-1} 2^\tau \cdot 2 \cdot 10(n-k-\eta+\tau+1) \\ &= 20(2^\eta(n-k-1) - (n-k-\eta-1)).\end{aligned} \tag{44}$$

For convenience, the factorisation complexity is analysed by also considering the process that realises the transform of $\widetilde{Q}_v(x, y)$ to $Q_v(x, y)$ as in (35). Let ω denote the number of coefficients of $\tilde{q}_{v,0}(x, y)$, then $\tilde{q}_{v,1}(x, y)$ has approximately $n-k+1-\omega$ coefficients. Since $V(x)$ has $k+1$ coefficients, $2\omega(k+1)$ finite field operations are required in realising $\tilde{q}_{v,0}(x, y) \cdot V(x)$. In factorising each polynomial, k message symbols need to be determined. Determining each message symbol requires a finite field operation and updating the polynomial requires $2(n-k+1-\omega)$ finite field operations [22, 24]. In the worst-case decoding event, 2^η interpolated polynomials need to be factorised and the factorisation complexity is

$$\begin{aligned}\mathcal{C}_{\text{fac}} &= 2^\eta \cdot 2\omega(k+1) + 2^\eta \cdot k(2(n-k+1-\omega)+1) \\ &= 2^{\eta+1}k(n-k+1) + 2^{\eta+1}\omega + 2^\eta k.\end{aligned} \tag{45}$$

By approximating $\omega = (n-k)/2$

$$\mathcal{C}_{\text{fac}} \simeq 2^{\eta+1}k(n-k+1) + 2^\eta n. \tag{46}$$

For the ML codeword validation, finite field operations are required to re-encode the message candidates. In the worst-case decoding event, there are 2^η message vectors to be encoded and the

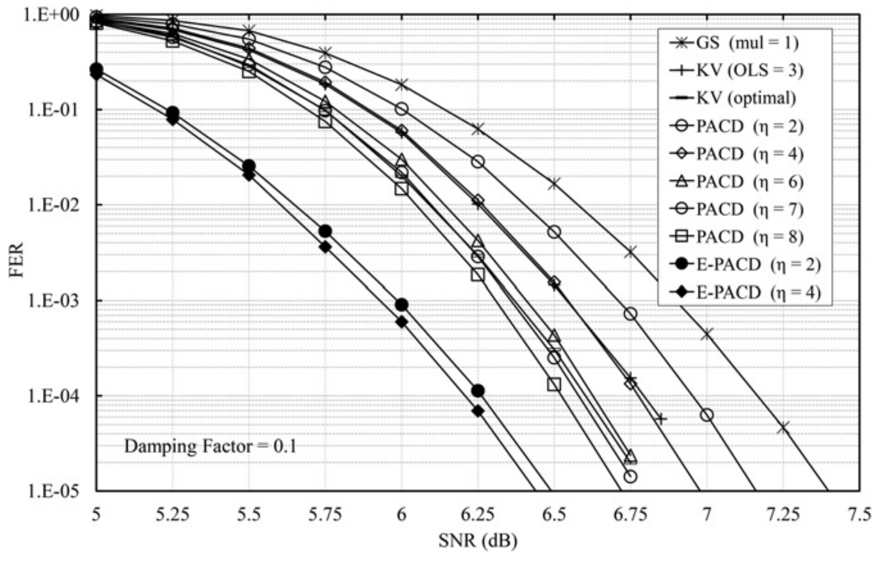

Fig. 7 *Performance of the (255, 239) RS code over the AWGN channel*

Table 1 Worst case complexity of the PACD algorithm in decoding the (15, 11) RS code

	Ana. ($\eta = 2$)	Sim. ($\eta = 2$)	Ana. ($\eta = 4$)	Sim. ($\eta = 4$)
$\mathcal{C}_{\text{int}}^{(1)}$	1992	1875	1992	1875
$\mathcal{C}_{\text{int}}^{(2)}$	30	33	0	0
$\mathcal{C}_{\text{int}}^{(3)}$	220	223	980	983
$\mathcal{C}_{\text{fac}}$	500	445	2000	1783
$\mathcal{C}_{\text{val}}$	1260	1203	5040	4813
$\mathcal{C}_{\text{PACD}}$	4002	3779	10,012	9454

Table 2 Worst case complexity of the PACD algorithm in decoding the (31, 27) RS code

	Ana. ($\eta = 2$)	Sim. ($\eta = 2$)	Ana. ($\eta = 4$)	Sim. ($\eta = 4$)
$\mathcal{C}_{\text{int}}^{(1)}$	22,600	22,214	22,600	22,214
$\mathcal{C}_{\text{int}}^{(2)}$	30	35	0	0
$\mathcal{C}_{\text{int}}^{(3)}$	220	239	980	1044
$\mathcal{C}_{\text{fac}}$	1204	1207	4816	4830
$\mathcal{C}_{\text{val}}$	6572	6385	26,288	25,553
$\mathcal{C}_{\text{PACD}}$	30,626	30,080	54,684	53,641

generation of each codeword symbol requires $2k - 1$ finite field operations. Hence, the ML validation complexity is

$$\mathcal{C}_{\text{val}} = 2^{\eta} \cdot n(2k - 1). \tag{47}$$

Summarising the above analysis, in the worst-case decoding event, the PACD algorithm's complexity can be approximated as

$$\mathcal{C}_{\text{PACD}} \simeq (k+1)^3 + 6k(n-k) + 2^{\eta+1}(k(2n-k+1) + 10(n-k-1)), \tag{48}$$

in which $\mathcal{C}_{\text{int}}^{(2)}$ and term $20(n-k-\eta+1)$ of $\mathcal{C}_{\text{int}}^{(3)}$ have been marginalised. Therefore, with a small η, the PACD complexity is dominated by the re-encoding. With increasing η, $2^{\eta+1}(k(2n-k+1) + 10(n-k-1))$ increases exponentially. The uncommon element interpolation, factorisation and the ML codeword validation become the dominant factor.

Tables 1 and 2 validate the above analysis from the simulation results that are obtained by decoding the (15, 11) and the (31, 27) RS codes, respectively. They are averaged over 1000 worst case decoding events. It can be seen that the simulation results match well with the analytical results, validating the above analysis. Note that for the two RS codes, when $\eta = 4$, $A_c = \emptyset$ which leads to $\mathcal{C}_{\text{int}}^{(2)} = 0$.

In the best case decoding event, only the first test-vector will be decoded. The uncommon element interpolation results in a complete *root-to-leaf* path of the binary tree. Therefore, the complexity of the common and the uncommon element interpolations can be merged as $\sum_{i=1}^{n-k} 10i = 5(n-k)(n-k+1)$. By considering only one test-vector to be processed, $\mathcal{C}_{\text{fac}}$ and $\mathcal{C}_{\text{val}}$ become $2k(n-k+1) + n$ and $n(2k-1)$, respectively. Therefore, in the best case decoding event, the PACD algorithm's complexity becomes

$$\mathcal{C}_{\text{PACD}} \simeq (k+1)^3 + 2k(2n-k+1) + (n-k)(5n+k+5), \tag{49}$$

and it is dominated by the re-encoding.

Simulation results on the average complexity in decoding the (15, 11) and the (31, 27) RS codes over the AWGN channel are shown in Figs. 8–10. Again, the complexity is averaged over 1000 decoding events per signal-to-noise ratio (SNR). In Figs. 8 and 9, the comparison benchmarks include the GS algorithm, the LCC and the BF-LCC algorithms both of which are considered as the predecessors of the PACD algorithm. They both show that by increasing the SNR, the average complexity of the PACD algorithm can be reduced significantly due to its progressive decoding mechanism. In contrast, the average complexity of the LCC and the BF-LCC algorithms is less sensitive to the channel condition, since they always terminate after decoding all the test-vectors. For the PACD algorithm, in the low SNR region (≤ 2 dB), the worst case decoding events dominate. Hence, its average complexity is similar to that of the LCC and the BF-LCC algorithms. While in the high SNR region (≥ 7 dB), the best case decoding events dominate. As a result, its average complexity converges to the minimal level, which is also the complexity of the GS algorithm. Note that in the best case decoding event, the PACD algorithm is less complex than the LCC algorithm with $\eta = 1$. This is because the PACD algorithm can always deliver the intended message by decoding the highest potential test-vector, while the LCC algorithm when $\eta = 1$ would have to decode two test-vectors. Moreover, with the same η value, the BF-LCC algorithm is slightly more complex than the LCC algorithm. This extra difference is incurred by the backward interpolation of the BF-LCC algorithm, in which the size of the polynomial is always greater than that of the LCC algorithm.

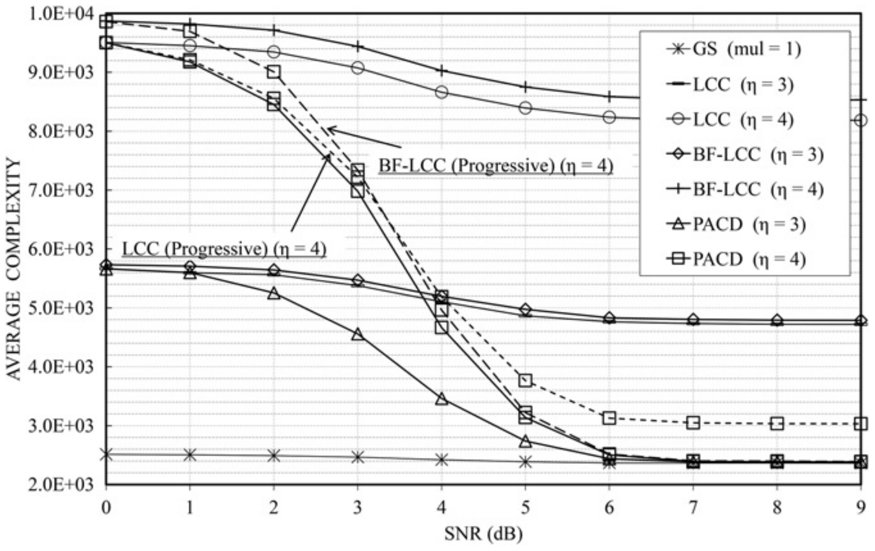

Fig. 8 *Average complexity in decoding the (15, 11) RS code using the PACD algorithm*

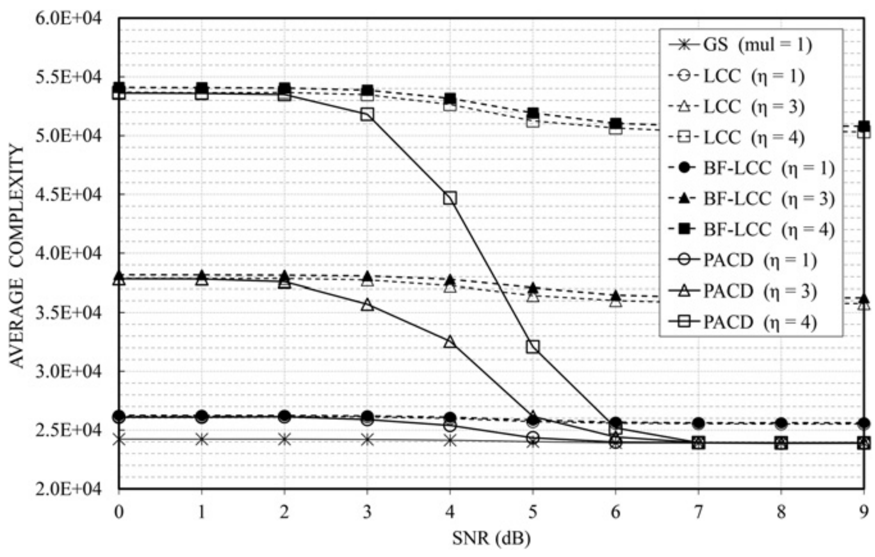

Fig. 9 *Average complexity in decoding the (31, 27) RS code using the PACD algorithm*

It should be acknowledged that both the LCC and the BF-LCC algorithms can be easily equipped with the progressive decoding feature. That means both algorithms can terminate their decoding once the intended message is found, resulting in a similar channel dependent complexity performance as the PACD algorithm. Fig. 8 further shows the complexity of the two benchmarks with the progressive decoding mechanism. For the progressive LCC and BF-LCC algorithms, it is ensured that the first test-vector to be decoded is the hard-decision received word, which is the same as the PACD algorithm. It shows the PACD algorithm remains the simplest compared with its predecessors. At the low SNR region in which the worst case decoding events dominate, the BF-LCC algorithm is more complex. This is triggered by the backward interpolation and larger size of the interpolated polynomials. In the opposite end of the SNR spectrum, the PACD and the BF-LCC algorithms have a similar complexity, since in most of the decoding events they both result in only one *root-to-leaf* path in the binary tree. However, the LCC algorithm still results in a full growth of the binary tree, bringing in a high computational cost. This comparison shows the merit of performing the progressive Chase decoding in a defined order. To further consolidate the claim, Table 3 shows the probability of yielding the ML codeword by decoding a particular ordered test-vector. It is measured over the AWGN channel with 100,000 decoding events per SNR. Be aware that there can be more than one test-vector that yields the ML codeword and all of them will be counted in the statistics. It shows that the reliability function of (10) can well identify the more potential test-vectors. It will result in an earlier termination of the Chase decoder.

Fig. 10 further sheds light on the complexity of the E-PACD algorithm in decoding the (15, 11) RS code. All the algorithms are assisted by the re-encoding transform. It can be seen that complexity of the KV algorithm also decreases with an increased SNR. This is because at high SNR, the multiplicity matrix [7] becomes sparser with greater non-zero entries. As a result, the re-encoding transform will be more capable in reducing the interpolation complexity. On the basis of Fig. 5 we know that with $\eta = 4$, both the PACD and the E-PACD algorithms can outperform the KV algorithm with an OLS of three, while Fig. 10 shows that they are less complex. In the low SNR region, the E-PACD algorithm is more complex than the PACD and the LCC algorithms. This is because the intended message often cannot be found after decoding all the first 2^{η} test-vectors and more PACD attempts are needed. However, note that though the E-PACD

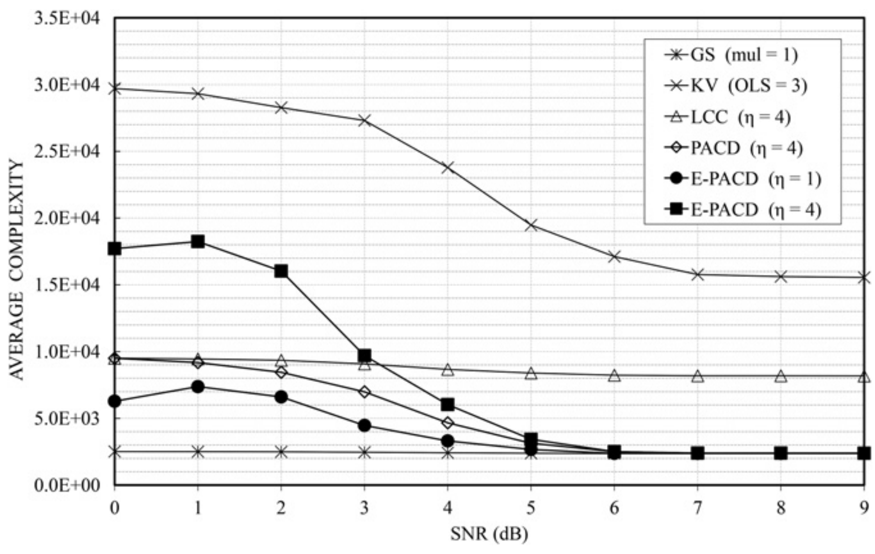

Fig. 10 *Average complexity comparison in decoding the (15, 11) RS code*

Table 3 Probability of the ordered test-vectors yielding an ML codeword for the (15, 11) RS code and $\eta = 3$

SNR, dB	$\mathcal{Y}'_{v_1}$	$\mathcal{Y}'_{v_2}$	$\mathcal{Y}'_{v_3}$	$\mathcal{Y}'_{v_4}$	$\mathcal{Y}'_{v_5}$	$\mathcal{Y}'_{v_6}$	$\mathcal{Y}'_{v_7}$	$\mathcal{Y}'_{v_8}$
3	0.364	0.329	0.315	0.296	0.268	0.246	0.221	0.171
4	0.661	0.621	0.595	0.569	0.525	0.488	0.430	0.317
5	0.888	0.862	0.839	0.811	0.776	0.728	0.634	0.398

algorithm functions with three ABP iterations, at the low SNR region, its average complexity is far less than four times of that of the PACD algorithm. This is because the redundant decodings have been eliminated by utilising the memorised interpolation information of each PACD attempt. Fig. 5 shows that the E-PACD algorithm with $\eta = 1$ outperforms the PACD algorithm with $\eta = 4$, while Fig. 10 also demonstrates that the former requires less finite field operations.

7 Memory analysis

This section further analyses the memory requirement of the PACD algorithm. The difference between the PACD algorithm and the LCC algorithm lies in the uncommon element interpolation, for which the PACD algorithm needs to memorise the intermediate interpolation information, i.e. the intermediate nodes of the binary tree. The memory requirement is analysed in the worst case decoding event.

The memory requirement is measured as the number of polynomial coefficients that need to be stored during the decoding. We assume that one coefficient is stored in one memory unit. Both the PACD and LCC algorithms initialise the polynomial set $\mathcal{G}$ as in (19), and each of its polynomials has one coefficient. We now analyse the memory requirement of both the LCC and the PACD algorithms and further compare them.

The LCC algorithm grows the binary tree in a *layer-by-layer* manner. It finally obtains 2^η leaves that correspond to 2^η polynomial sets. Before reaching the leaves, the LCC algorithm needs to memorise all the nodes of layer $\eta - 1$. Since the polynomials in the layer are obtained by interpolating $n - k - 1$ points, storing each polynomial requires at most $2(n - k)$ memory units. With two polynomials in each set, the memory requirement of the LCC algorithm is

$$\mathfrak{M}_{\text{LCC}} = 2^{\eta-1} \cdot 4(n-k). \tag{50}$$

In contrast, the PACD algorithm grows the binary tree in a *depth-first-search* manner. Instead of storing all 2^η leaves, it needs to store all the intermediate nodes of the binary tree, i.e. $\mathcal{G}_v^{(\tau)}$ with $0 \leqslant \tau < \eta$, and the leaf that corresponds to the current test-vector. The polynomial set of the root is obtained by interpolating $n - k - \eta$ common points. Therefore, storing the root requires $2(n - k - \eta + 1)$ memory units. After interpolating τ points in the uncommon element interpolation, storing a polynomial set that is a node at layer τ requires $4(n - k - \eta + 1 + \tau)$ memory units. Moreover, storing the leaf that corresponds to the current test-vector requires $4(n - k + 1)$ memory units. Therefore, the memory requirement of the PACD algorithm is

$$\begin{aligned}\mathfrak{M}_{\text{PACD}} &= \sum_{\tau=0}^{\eta-1} 2^\tau \cdot 4(n-k-\eta+1+\tau) + 4(n-k+1) \\ &= 4(2^\eta(n-k-1) + (\eta+2)).\end{aligned} \tag{51}$$

Comparing $\mathfrak{M}_{\text{LCC}}$ of (50) and $\mathfrak{M}_{\text{PACD}}$ of (51) yields a discrepancy of

$$\mathfrak{M}_{\text{LCC}} - \mathfrak{M}_{\text{PACD}} = 4(2^{\eta-1}(n-k-2) + \eta + 2). \tag{52}$$

It is obvious that $\mathfrak{M}_{\text{LCC}} - \mathfrak{M}_{\text{PACD}} > 0$, $\forall \eta$. Equation (52) shows that the discrepancy grows with increased η. It will be more expensive to store all the leaves than all the intermediate nodes of the binary tree. However, it should be pointed out that for the LCC algorithm, the stored polynomial set can be erased once it has been factorised. While for the PACD algorithm, the intermediate nodes have to be stored until the decoding terminates. Therefore, the PACD algorithm demands a longer occupation of the assigned memory. Nonetheless, memory requirement will be an inevitable cost in realising progressive decoding [17].

8 Conclusion

This paper has proposed the PACD algorithm and an enhanced variant for RS codes. In the PACD algorithm, each test-vector will be sequentially decoded according to its potential for yielding the intended message. Progressive decoding will be terminated once the intended message has been found. By examining the similarity between the decoded test-vectors and the current one, complexity of decoding the current test-vector can be reduced by fully utilising the existing interpolation information. Furthermore, the E-PACD algorithm has been introduced by coupling the PACD algorithm with the ABP decoding that is capable to generate new test-vectors by enhancing the received information. As a result, stronger algebraic Chase decoding performance can be obtained without increasing the decoding complexity exponentially. Complexity analysis of the PACD algorithm has been conducted to formulate its computational cost. Complexities of the worst case and the best case decoding events have been analysed, bounding the actual complexity of a PACD event. Together with the performance analysis, it has been shown that the PACD algorithm can outperform both the GS and the KV algorithms with less computational cost. Assisted by the ABP decoding, the E-PACD algorithm significantly outperforms the PACD algorithm without incurring an exponentially increased decoding complexity. Memory analysis of the PACD algorithm has also been performed demonstrating the cost of storage requirement in realising the progressive decoding.

9 Acknowledgments

This work is sponsored by the National Natural Science Foundation of China (NSFC) with project ID 61372079, the National Basic Research Program of China (973 program) with project Number 2012CB316100 and the Fundamental Research Funds for the Central Universities in China.

10 References

1 Reed, I.S., Solomon, G.: 'Polynomial codes over certain finite fields', *J. Soc. Ind. Appl. Math.*, 1960, **8**, pp. 300–304
2 Massey, J.L.: 'Shift register synthesis and BCH decoding', *IEEE Trans. Inf. Theory*, 1969, **15**, (1), pp. 122–127
3 Welch, L., Berlekamp, E.R.: 'Error correction for algebraic block codes'. Proc. IEEE Int. Symp. Information Theory (ISIT), St. Jovite, Canada, September 1983
4 Sorger, U.: 'A new Reed–Solomon code decoding algorithm based on Newton's interpolation', *IEEE Trans. Inf. Theory*, 1993, **39**, (2), pp. 358–365
5 Sudan, M.: 'Decoding of Reed–Solomon codes beyond the error-correction bound', *J. Complex.*, 1997, **13**, (1), pp. 180–193
6 Guruswami, V., Sudan, M.: 'Improved decoding of Reed–Solomon and algebraic-geometric codes', *IEEE Trans. Inf. Theory*, 1999, **45**, (6), pp. 1757–1767
7 Koetter, R., Vardy, A.: 'Algebraic soft-decision decoding of Reed–Solomon codes', *IEEE Trans. Inf. Theory*, 2003, **49**, (11), pp. 2809–2825
8 Chen, L., Carrasco, R.A., Chester, E.G.: 'Performance of Reed–Solomon codes using the Guruswami-Sudan algorithm with improved interpolation efficiency', *IET Commun..*, 2007, **1**, (2), pp. 241–250
9 McEliece, R.J.: 'The Guruswami-Sudan decoding algorithm for Reed–Solomon codes'. IPN Progress Report, 42-153, May 2003

10 Wu, Y.: 'New list decoding algorithms for Reed–Solomon and BCH codes', *IEEE Trans. Inf. Theory*, 2008, **54**, (8), pp. 3611–3630
11 Koetter, R., Ma, J., Vardy, A.: 'The re-encoding transformation in algebraic list-decoding of Reed–Solomon codes', *IEEE Trans. Inf. Theory*, 2011, **57**, (2), pp. 633–647
12 Bellorado, J., Kavcic, A.: 'A low-complexity method for chase-type decoding of Reed–Solomon codes'. Proc. IEEE Int. Symp. Information Theory (ISIT), Seattle, WA, July 2006
13 Bellorado, J., Kavcic, A.: 'Low-complexity soft-decoding algorithms for Reed–Solomon codes – part I: an algebraic soft-in hard-out chase decoder', *IEEE Trans. Inf. Theory*, 2010, **56**, (3), pp. 945–959
14 Zhu, J., Zhang, X., Wang, Z.: 'Backward interpolation architecture for algebraic soft-decision Reed–Solomon decoding', *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2009, **17**, (11), pp. 1602–1615
15 Zhang, X., Zhu, J.: 'Algebraic soft-decision decoder architectures for long Reed–Solomon codes', *IEEE Trans. Circuits Syst. II*, 2010, **57**, (10), pp. 787–792
16 Tang, S., Ma, X.: 'A new Chase-type soft-decision decoding algorithm for Reed–Solomon codes', Available at http://www.arxiv.org/abs/1309.1555
17 Chen, L., Tang, S., Ma, X.: 'Progressive algebraic soft-decision decoding of Reed–Solomon codes', *IEEE Trans. Commun.*, 2013, **61**, (2), pp. 433–442
18 Cassuto, Y., Bruck, J., McEliece, R.J.: 'On the average complexity of Reed–Solomon list decoders', *IEEE Trans. Inf. Theory*, 2013, **59**, (4), pp. 2336–2351
19 Kaneko, T., Nishijima, T., Inazumi, H., *et al.*: 'An efficient maximum-likelihood-decoding algorithm for linear block codes with algebraic decoder', *IEEE Trans. Inf. Theory*, 1994, **40**, (2), pp. 320–327
20 El-Khamy, M., McEliece, R.J.: 'Iterative algebraic soft-decision list decoding of Reed–Solomon codes', *IEEE J. Sel. Areas Commun.*, 2006, **24**, (3), pp. 481–490
21 Jiang, J., Narayanan, K.R.: 'Iterative soft-input soft-output decoding of Reed–Solomon codes by adapting the parity-check matrix', *IEEE Trans. Inf. Theory*, 2006, **52**, (8), pp. 3746–3756
22 Roth, R., Ruckenstein, G.: 'Efficient decoding of Reed–Solomon codes beyond half the minimum distance', *IEEE Trans. Inf. Theory*, 2000, **46**, (1), pp. 246–257
23 El-Khamy, M., McEliece, R.J.: 'Bounds on the average binary minimum distance and the maximum likelihood performance of Reed–Solomon codes'. Proc. the 42nd Allerton Conf. on Communication, Control and Computing, Monticello, USA, October 2006
24 Chen, L., Carrasco, R.A., Johnston, M., *et al.*: 'Efficient factorisation algorithm for list decoding algebraic-geometric and Reed–Solomon codes'. Proc. IEEE Int. Communications Conf. (ICC), Glasgow, UK, May 2007