On the Average Complexity of Reed-Solomon List Decoders

Yuval Cassuto, *Member, IEEE,* Jehoshua Bruck, *Fellow, IEEE,* and Robert J. McEliece, *Life Fellow, IEEE*

***Abstract*—The number of monomials required to interpolate a received word in an algebraic list decoder for Reed-Solomon codes depends on the instantaneous channel error, and not only on the decoder design parameters. The implications of this fact are that the decoder should be able to exhibit lower decoding complexity for low-weight errors, and consequently enjoy a better average-case decoding complexity and a higher decoding throughput. On the analytical side, this paper studies the dependence of interpolation costs on instantaneous errors, in both hard- and soft-decision decoders. On the algorithmic side, it provides an efficient interpolation algorithm, based on the state-of-the-art interpolation algorithm, that enjoys reduced running times for reduced interpolation costs.**

***Index Terms*—Algebraic list decoding, bivariate interpolation, Kötter-Vardy soft decoding, Reed-Solomon codes.**

I. Introduction

THE core precept of coding theory is the trade-off between redundancy and correction capability. Countless constructions and bounds tie codes' correction capabilities with the corresponding redundancy that they carry. Outside this main front of coding theory stands the field of list decoding, debuted in the late 1950s with two articles by Elias [5] and by Wozencraft [21]. In list decoding, both the code redundancy and its (worst case) correction capability are fixed, and the decodability of the codes is examined for increasing decoding radii, beyond the unique-decoding bound. The field of algorithmic list decoding, whereby *efficient* decoders decode codes beyond their unique-decoding radius, was born in 1997 when Sudan introduced a polynomial-time list decoder for Reed-Solomon (RS) codes [17]. Coming from the theoretical computer science community, the great impact of this result initially came from its applicability to a multitude of open problems in complexity theory (see [18], [8] for survey papers). But not surprisingly, the advent of efficient list decoding of RS codes quickly engaged the information theory community, interested in the more practical facets of algorithmic list decoding. In particular, information theory researchers sought to refine the "polynomial time" decodability into a workable decoder, applicable to practical information-processing systems. Their starting point was an algorithm, which although polynomial time, has a prohibitive complexity for all but very short or very low-rate codes. Therefore, the research focus settled on the following two directions.

- **algorithmic efficiency**, i.e., improving the (polynomial time) complexity of bi-variate interpolation and factorization, the two main building blocks of the list-decoding algorithm.
- **soft decoding**, i.e., using soft decoding to achieve decoding gains even for codes whose parameters enjoy no decoding improvement by the original (hard decision) list decoder (or those that do improve, but only with prohibitive complexity).

The key contributions in the algorithmic efficiency direction are the interpolation algorithm attributed to Kötter [11], described in [14], and the factorization algorithm of Roth-Ruckenstein [16]. The soft-decoding direction was opened by Kötter and Vardy in [12]. The present paper further continues the refinement and improvement of list decoding algorithms. For codes that already qualify to be (soft or hard) list decoded with feasible complexity, this paper aims at studying and improving their decoding average-case complexity – contributing to higher decoding throughput in real-life decoders. While the worst-case list-decoding complexity of RS codes is well understood as a function of the code parameters and the decoding radius, this worst-case complexity analysis ignores the effect of the number of *instantaneous errors* on the decoding complexity. The mere observation that low-error channel outputs require fewer interpolation monomials than high-error ones is all but obvious. However, in many systems that employ RS codes, the average number of instantaneous errors introduced by the channel is typically much lower than the decoder's worst-case decoding radius. Hence differentiating the decoding complexity based on the number of instantaneous errors significantly improves the average decoding time, and in turn the decoder's throughput. When a single decoder is implemented using synchronous hardware, then admittedly it is the worst-case running time that determines the system performance. However, in many cases in practice, multiple decoders serve – in parallel – a queue of incoming received words, and in such cases lowering the average running time can have a significant impact on the system's performance or cost. Saving decoding operations can also contribute to reduction of the decoder's power consumption. An average-case analysis of classical RS decoders is pursued by Berlekamp and McEliece in [2], where the running-time dependence on the error weight is obtained experimentally. Berlekamp, a pioneer in hardware implementation of algebraic decoders (more known of course

Yuval Cassuto is with the Department of Electrical Engineering, Technion – Israel Institute of Technology, Haifa 32000, Israel (e-mail: ycassuto@ee.technion.ac.il).

Jehoshua Bruck is with the Department of Electrical Engineering, California Institute of Technology, 1200 E. California Blvd., Mail Code 136-93, Pasadena, CA 91125, U.S.A. (e-mail: bruck@paradise.caltech.edu).

Robert J. McEliece is with the Department of Electrical Engineering, California Institute of Technology, 1200 E. California Blvd., Mail Code 136-93, Pasadena, CA 91125, U.S.A. (e-mail: rjm@systems.caltech.edu).

This work was supported in part by the EU Marie Curie Career Integration Grant, by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI), and by the Caltech Lee Center for Advanced Networking. Part of the material in this paper was presented at the 2005 IEEE International Symposium on Communication Theory and Applications (ISCTA), Ambleside UK.

as a prominent mathematician), observed with McEliece in [2] that the gap between worst-case and average-case running times can be utilized for higher throughput decoders employing a buffered architecture. This method and its accompanying analysis tools apply to the results of this present paper as well. The study of [7] seeks to improve the average case complexity of the RS algebraic soft decision decoder [12] by using a layered decoder whose decoding time depends on the instantaneous channel noise. In this paper, the average-case analysis hinges on the dependence of the *interpolation cost* (the number of required interpolation monomials) on the error weight. This dependence is studied and quantified in Section III using analytical tools for hard-decision decoders. The tightness of the proved bounds is verified experimentally by providing interpolation results of a sample code. Soft-decision list decoding is the subject of Section IV, where instantaneous interpolation costs are bounded for soft-decision decoders as a function of the code and decoder's design parameters, as well as the channel statistics. For discrete-input additive-noise channels with known noise statistics, the analysis gives upper bounds on soft-decision interpolation costs that apply to any error vector with a given number of hard errors. When the noise wraps around from the last discrete symbol back to the first one, symmetry properties make these bounds especially simple to obtain. Later in Section IV, a more direct method to bound the *average* interpolation cost is given (without using bounds on the instantaneous interpolation costs). This result provides an analytical tool to predict the average-case complexity of the soft-decision decoder, as promised by the paper title. The section on soft decoding continues with an experimental comparison of the instantaneous interpolation costs of hard-decision vs. soft-decision decoders. The comparison reveals quite a different average-case behavior of the two decoders, even when their worst-case complexity is identical. Then, in Section V, an interpolation algorithm is proposed whose running time favorably depends on the instantaneous interpolation cost. This algorithmic proposition achieves improved average-case running time for both hard-decision and soft-decision decoding. At the end of Section V, the complexity savings of the new algorithm are demonstrated on decoder implementations for the $[31, 15]$ and $[127, 60]$ RS codes. For the $[127, 60]$ code, savings factors of up to 2-3 are observed, multiplied by millions of arithmetic operations consumed by the interpolation. Therefore, implementing the proposed algorithm is extremely attractive in cases where a low-rate, high correction capability code is used to cover rare high-order error events, while most of the decoding instances have much lower error counts. It is important to emphasize that the gains in the average-case running times come with no sacrifice to the worst-case decoding time of the standard GS decoder.

II. Review of Guruswami-Sudan Algebraic List Decoding

A review of algebraic list-decoding algorithms is now given as an aid to the forthcoming study of list-decoding complexity. A codeword C from an $[n, k, d]$ Reed-Solomon (RS) code is the evaluation of a degree $k-1$ or less message polynomial $f(x)$ on n distinct points of GF(q), $\{\alpha_1, \ldots, \alpha_n\}$. The prime power q is the alphabet size of the code. Let E be an error vector of Hamming weight e over the same alphabet GF(q). The received word R is defined as $R = C + E$, over GF(q) arithmetic. Classical decoding algorithms of RS codes, e.g., the Berlekamp algorithm, the Massey algorithm, and their predecessor Peterson-Gorenstein-Zierler algorithm (see, e.g., [3] for a description of these algorithms), all attempt to efficiently solve the following linear system of ν equations (ν = number of errors):

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{\nu-1} & S_\nu \\ S_2 & S_3 & \cdots & S_\nu & S_{\nu+1} \\ S_3 & S_4 & \cdots & S_{\nu+1} & S_{\nu+2} \\ \vdots & & & & \vdots \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-2} & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \Lambda_{\nu-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ -S_{\nu+3} \\ \vdots \\ -S_{2\nu} \end{bmatrix}$$

where Λ_i are the coefficients of the unknown error-locator polynomial and S_j are the known syndromes. When $d \geqslant 2\nu + 1$, this system of equations has a unique solution and thus the algorithms mentioned above can decode errors up to half the minimum distance: the unique-decoding bound.

A completely different approach to decoding RS codes that can correct more errors than classical algorithms, has been introduced by Sudan [17], and improved by Guruswami and Sudan [9], using relatively simple but powerful algebraic-geometric ideas. In the Guruswami-Sudan (GS) algorithm [9], the received word is used to interpolate a bivariate polynomial $Q(x, y)$, called below the *interpolation polynomial*. To achieve a large correction radius, an integer decoder parameter m, called the *interpolation multiplicity* is chosen, and $Q(x, y)$ is taken to be the minimal $(1, k-1)$-weighted degree[1] bivariate polynomial that satisfies the following $n\binom{m+1}{2}$ constraints: $D_{r,s}Q(\alpha_j, R_j) = 0$ for $j = \{1, \ldots, n\}$ and r, s such that $r + s < m$. R_j is the received symbol at code-coordinate j. The expression $D_{r,s}Q(\alpha, \beta)$ refers to the Hasse derivative of x-order r and y-order s, evaluated at the point $x = \alpha, y = \beta$ (more on Hasse derivatives in Section III-A below and in [14].) If these interpolation constraints are satisfied by $Q(x, y)$, it is guaranteed that codewords within the prescribed decoding radius of the decoder will be found by factorization of $Q(x, y)$. A block diagram of the GS list-decoding algorithm is given in Figure 1 below.

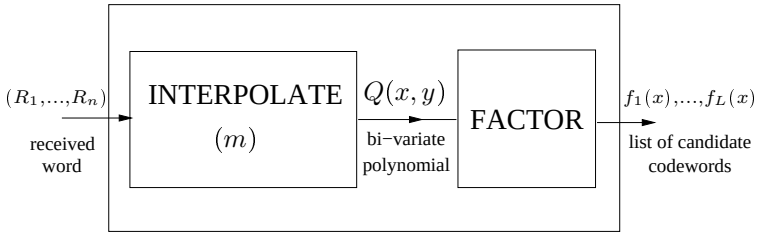

Figure 1. Block diagram of the Guruswami-Sudan list-decoding algorithm.

[1]The (u, v)-weighted degree of a bivariate polynomial is the maximum of $ui + vj$ over all of its monomials $x^i y^j$.

By formulating the interpolation as a system of homogeneous linear equations, it has been observed in [9] that $n\binom{m+1}{2}+1$ coefficients are sufficient to make $Q(x,y)$ satisfy the above constraints. We denote by $\mathbb{C}_{\rm wc}$ this worst case number of interpolation coefficients, so $\mathbb{C}_{\rm wc} = n\binom{m+1}{2}+1$. $\mathbb{C}_{\rm wc}$ will be later called the $w$orst $c$ase *interpolation cost* of the GS (n,k,m) decoder. The key yield from that decoding scheme is that a sufficient condition to correct e errors is $m(n-e) > d_{1,k-1}(\mathbb{C}_{\rm wc})$, where $d_{u,v}(J)$ is the minimal (u,v)-weighted degree of a bivariate polynomial with J coefficients. Since in general the number of correctable errors e is larger than $\lfloor (d-1)/2 \rfloor$, i.e., half the minimum distance of the code, the decoder output is a *list* that possibly contains multiple codewords. Hence the qualifier *list-decoding* is used for the GS decoder, as well as for other decoders that correct beyond the unique decoding bound $\lfloor (d-1)/2 \rfloor$. Throughout the paper, we assume that the monomials of the interpolation polynomials are ordered by non-decreasing $(1,k-1)$-weighted degrees, with reverse-lexicographic tie-breaking, i.e. $x^{(k-1)t}$ precedes $x^{(k-1)(t-1)}y$ (or in general, if two monomials have the same $(1,k-1)$-weighted degree, then a monomial with lower y-degree precedes others with higher y-degrees). We refer to this ordering as the *standard monomial order*. For a more detailed discussion of multivariate polynomials and monomial ordering please refer to [4].

III. Interpolation Polynomials for Low-Weight Errors

In this section we develop tools for bounding the interpolation cost given the instantaneous errors from the channel. The channel model we adopt in this section is the standard hard-error block model. This refers to a channel with inputs and outputs from a certain finite field, where a subset of the symbols from the length-n code block may change to different (erroneous) symbols. The number of symbols in this erroneous subset is called the *error weight*, denoted e (no restriction is made on the locations of the errors within the block.)

The bounds are obtained by providing classes of interpolation polynomials for received words resulting from an error of a given weight, and then analyzing the degrees of these polynomials to get upper bounds on interpolation costs. An obvious outcome from upper bounds on the interpolation cost is reduced complexity of the factorization step, owing to smaller input sizes. A less obvious, but much more computationally significant outcome, is the possibility to devise a new interpolation algorithm that runs with complexity proportional to the instantaneous interpolation cost, without having to know the received-error weight. Such an algorithm is presented in Section V. To this end, we have introduced the worst case interpolation cost $\mathbb{C}_{\rm wc}$, which is determined by the decoder parameters n,k,m. For the sake of the forthcoming analysis, we define the error-weight dependent interpolation cost $\mathbb{C}_e(\epsilon)$ as the number of interpolation coefficients required given an error word ϵ of Hamming weight e. Note that $\mathbb{C}_e(\epsilon)$ is not a function of e alone; different interpolation costs are possible for different error words of a given weight e. Therefore, we will seek upper bounds on $\mathbb{C}_e(\epsilon)$ that apply to *all* error words ϵ of weight e. We clarify that the analysis of interpolation costs to follow is not restricted to numbers of errors beyond the unique-decoding bound, but rather to the full spectrum of error counts. In fact, some of the bounds on interpolation costs will only be useful for error counts within the unique-decoding bound. Since the scope of this paper is to improve the average-case running time of the decoder, which in many applications may be dominated by low-error instances, these bounds are indeed effective analysis tools.

A. Hasse derivatives

Because of their central role in the interpolation procedure, Hasse derivatives and their properties are discussed in detail.

Definition 1. (The (r,s) Hasse derivative) *The (r,s) Hasse derivative of a polynomial $Q(x,y)$, denoted $D_{r,s}Q(x,y)$, is defined for any integer pair $r \geqslant 0$, $s \geqslant 0$ as*

$$D_{r,s}Q(x,y) = \sum_{i,j} \binom{i}{r}\binom{j}{s} a_{i,j} x^{i-r} y^{j-s}.$$

where $a_{i,j}$ is the coefficient of $x^i y^j$ in $Q(x,y)$.

Hasse derivatives owe their use in RS list-decoding to the following fact

$$D_{r,s}Q(\alpha,\beta) = \mathrm{coeff}_{x^r y^s} Q(x+\alpha, y+\beta).$$

In words, the coefficient of $x^r y^s$ in the polynomial $Q(x+\alpha, y+\beta)$ equals the (r,s) Hasse derivative of the polynomial $Q(x,y)$, evaluated at the point $x=\alpha, y=\beta$. We now turn to state (without proof) the well-known product rule for Hasse derivatives.

Lemma 2. (The Hasse derivative product rule) *For any integer pair $r \geqslant 0$, $s \geqslant 0$, the Hasse derivative of a product of L polynomials is decomposed as*

$$D_{r,s}\left[\prod_{i=1}^{L} Q_i(x,y)\right] = \sum_{\substack{r_1+\cdots+r_L=r \\ s_1+\cdots+s_L=s}} \prod_{i=1}^{L} D_{r_i,s_i} Q_i(x,y),$$

where $r_i \geqslant 0$, $s_i \geqslant 0$ are arbitrary integers that satisfy the sum condition on the right-hand side.

From Lemma 2 we get the following lemma.

Lemma 3. *Let m be a positive integer. If $Q(\alpha,\beta)=0$, then for every pair of non-negative integers r,s such that $r+s<m$, we have $D_{r,s}\left[Q(\alpha,\beta)^m\right]=0$, where $D_{r,s}\left[Q(\alpha,\beta)^m\right]$ is the r,s Hasse derivative of $Q(x,y)^m$, evaluated at (α,β).*

Proof: Lemma 2 states that

$$D_{r,s}\left[Q(x,y)^m\right] = \sum_{\substack{r_1+\cdots+r_m=r \\ s_1+\cdots+s_m=s}} \prod_{i=1}^{m} D_{r_i,s_i} Q(x,y).$$

Since $r+s = \sum_{i=1}^{m}(r_i+s_i) < m$, for every assignment to $r_1,s_1,\ldots,r_m,s_m$ at least one of the pairs (r_i,s_i) equals $(0,0)$. That means that every product in the sum contains at least one factor $D_{0,0}Q(x,y)=Q(x,y)$. Substituting $(x,y)=(\alpha,\beta)$, the right-hand side evaluates to zero. □

In the remainder of the section, m *will denote the interpolation multiplicity of the decoder*, as defined in Section II.

B. Closed form upper bound on interpolation costs

Theorem 4. *Let E be an error vector of weight e and let $\{j_1, j_2, \ldots, j_e\}$ be the error locations. Then there exists an interpolation polynomial whose last monomial, according to the standard monomial order, is $x^{em}y^m$. This polynomial can be explicitly written as*

$$Q(x,y) = \left[(y - f(x))(x - \alpha_{j_1})(x - \alpha_{j_2}) \cdots (x - \alpha_{j_e})\right]^m. \tag{1}$$

Proof: The polynomial

$$q(x,y) \triangleq (y - f(x))(x - \alpha_{j_1})(x - \alpha_{j_2}) \cdots (x - \alpha_{j_e})$$

evaluates to zero at each of the received points $(x,y) = (\alpha_i, R_i)$ (both the correct and corrupted symbols). By Lemma 3 we get that $Q(x,y) = q(x,y)^m$ satisfies $D_{r,s}Q(x,y) = 0$ for all $r + s < m$ and all $(x,y) = (\alpha_i, R_i)$, as needed for interpolation with multiplicity m. □

The strength of the arguments used in the proof above is that they allow to predict the *form* of interpolation polynomials for any error weight, even without constructively interpolating particular received words. We now state a lemma that will be used to translate interpolation-polynomial specifications such as (1) into upper bounds on the interpolation cost. Similar lemmas that connect polynomial degrees and costs have appeared before – e.g., in [12] – but we include Lemma 5 here with the current notation to offer a self-contained presentation.

Lemma 5. *Let $\phi(x,y)$ be the last monomial of a polynomial $Q(x,y)$, according to the standard monomial order. If $\phi(x,y)$ satisfies $d_{1,k-1}(\phi(x,y)) = \Delta$ and $d_{0,1}(\phi(x,y)) = \ell$, then the number J, of monomials in $Q(x,y)$, equals:*

$$J = (t+1)\left(\frac{1}{2}t(k-1) + u\right) + \ell + 1 \tag{2}$$

where t and u are, respectively, the quotient and residue of Δ when divided by $k-1$.

Proof: Given the standard monomial order, $Q(x,y)$ has all the monomials of $(1,k-1)$-weighted degree *less than* Δ, and $\ell + 1$ monomials with degree *exactly* Δ (those with y-degrees $\{0, 1, \ldots, \ell\}$). Summing both groups we get

$$\begin{aligned} J &= \sum_{j=0}^{t} \left[(k-1)(t-j) + u\right] + \ell + 1 \\ &= (t+1)\left(\frac{1}{2}t(k-1) + u\right) + \ell + 1. \end{aligned}$$

□

Now we are ready to combine Theorem 4 and Lemma 5 into an upper bound on the interpolation cost $\mathbb{C}_e(\epsilon)$, as a function of the instantaneous error weight e, as well as the code and decoder parameters k, m.

Corollary 6. *Let $m(e+k-1) = (k-1)t + u$, where t and $u < k-1$ are integers (t and u are, respectively, the quotient and residue of $m(e+k-1)$ when divided by $k-1$). For any error of weight e we have the following bound*

$$\mathbb{C}_e(\epsilon) \leqslant (t+1)\left(\frac{1}{2}t(k-1) + u\right) + m + 1. \tag{3}$$

Proof: Theorem 4 proves that there exists an interpolation polynomial whose last monomial is $x^{em}y^m$, which satisfies $d_{1,k-1}(x^{em}y^m) = m(e+k-1)$ and $d_{0,1}(x^{em}y^m) = m$. Substituting $\Delta = m(e+k-1)$ and $\ell = m$ in Lemma 5, we obtain the right-hand side of (3). The inequality in (3) comes from the fact that there *may* be other interpolation polynomials, besides the one of Theorem 4, with lower interpolation cost. □

To obtain a closed-form comparison of the bound on $\mathbb{C}_e$ in (3) to $\mathbb{C}_{\text{wc}}$, we first note a simpler, though less tight bound. Given a polynomial $Q(x,y)$ with $(1,k-1)$-weighted degree Δ, the number J of monomials in Q is bounded by [14]

$$J \leqslant \frac{\left(\Delta + 1 + \frac{k-1}{2}\right)^2}{2(k-1)}.$$

Substituting $\Delta = m(e+k-1)$, we get (for $e > 1$)

$$\mathbb{C}_e \leqslant \frac{\left[\left(m + \frac{1}{2}\right)(e+k-1)\right]^2}{2(k-1)}.$$

Now taking the ratio with $\mathbb{C}_{\text{wc}}$

$$\frac{\mathbb{C}_e}{\mathbb{C}_{\text{wc}}} \leqslant \frac{\left[\left(m + \frac{1}{2}\right)(e+k-1)\right]^2}{(k-1)m(m+1)n}.$$

Approximating $(m+1/2)^2 \approx m(m+1)$ and $(k-1)/n \approx k/n$, we get

$$\frac{\mathbb{C}_e}{\mathbb{C}_{\text{wc}}} \lessapprox \left[\frac{e}{n} + \frac{k}{n}\right]^2 \frac{n}{k},$$

and after defining $\rho = k/n$ to be the rate of the code and $\hat{e} = e/n$ to be the fractional number of errors, we get

$$\frac{\mathbb{C}_e}{\mathbb{C}_{\text{wc}}} \lessapprox \frac{\left[\hat{e} + \rho\right]^2}{\rho}. \tag{4}$$

An important special case, the no-error case, is when $\hat{e} = 0$, in which case (4) gives

$$\mathbb{C}_0 \lessapprox \rho \mathbb{C}_{\text{wc}}.$$

A direct proof (without approximation) of the bound on $\mathbb{C}_0$ is provided in Appendix B.

C. Tighter bounds for higher weight errors

When e is large, bounds on the interpolation cost can still be obtained, though using (3) may not be the best choice. For such cases we can use the following theorem.

Theorem 7. *Let E be an error vector of weight e and let $\{j_1, j_2, \ldots, j_e\}$ be the error locations. For any integer $0 \leqslant m' \leqslant m$, a polynomial of the form*

$$Q(x,y) = (y - f(x))^{m'} P(x,y)$$

is an interpolation polynomial when $P(x,y)$ suffices to satisfy at most $e\binom{m+1}{2} + (n-e)\binom{m-m'+1}{2}$ interpolation constraints.

Proof: We first find a minimal $(1,k$–$1)$-degree polynomial $P(x,y)$ that satisfies the following constraints. For the e corrupted locations $\{j_1, j_2, \ldots, j_e\}$, we require the usual interpolation constraints $D_{r,s}P(\alpha_{j_i}, R_{j_i}) = 0$, $i = 1, \ldots, e$, for every pair (r,s) such that $r + s < m$. For the $n-e$ uncorrupted

locations we require fewer such constraints: $D_{r,s}P(\alpha_j, R_j) = 0$ for every (r,s) such that $r+s < m-m'$. Since for the corrupted symbols the polynomial $P(x,y)$ alone satisfies all interpolation requirements, $Q(x,y)$ obviously does so too. As for the uncorrupted symbols we write

$$D_{r,s}Q(x,y) = \sum_{\substack{r_1+r_2=r \\ s_1+s_2=s}} D_{r_1,s_1}(y-f(x))^{m'} D_{r_2,s_2}P(x,y).$$

Splitting the sum into two disjoint intervals

$$\begin{aligned} D_{r,s}Q(x,y) = & \\ &= \sum_{\substack{r_1+r_2=r \\ s_1+s_2=s\ : \\ r_1+s_1<m'}} D_{r_1,s_1}(y-f(x))^{m'} D_{r_2,s_2}P(x,y) \\ &\quad + \sum_{\substack{r_1+r_2=r \\ s_1+s_2=s\ : \\ r_1+s_1\geqslant m'}} D_{r_1,s_1}(y-f(x))^{m'} D_{r_2,s_2}P(x,y) \\ &= \sum_{\substack{r_1+r_2=r \\ s_1+s_2=s\ : \\ r_1+s_1<m'}} D_{r_1,s_1}(y-f(x))^{m'} D_{r_2,s_2}P(x,y) \quad (5) \\ &\quad + \sum_{\substack{r_1+r_2=r \\ s_1+s_2=s\ : \\ r_2+s_2\leqslant r+s-m'}} D_{r_1,s_1}(y-f(x))^{m'} D_{r_2,s_2}P(x,y). \quad (6) \end{aligned}$$

When evaluated at $x=\alpha_j$, $y=R_j$ of the corrupted locations, by definition of $P(x,y)$ both (5) and (6) are zero. When evaluated at $x=\alpha_j$, $y=R_j$ of the uncorrupted locations, the sum in (5) is zero by Lemma 3, and the sum in (6) is zero since $r+s-m' < m-m'$, and the (r_2,s_2) Hasse derivatives of $P(x,y)$ such that $r_2+s_2 < m-m'$ vanish on the uncorrupted locations by construction. □

Corollary 8. *For any $0 \leqslant m' \leqslant m$, let $x^{d_x}y^{d_y}$ be the monomial whose index in the monomial order is $e\binom{m+1}{2} + (n-e)\binom{m-m'+1}{2}$. Further, let $d_x = q_x(k-1)+r_x$, where q_x and $r_x < k-1$ are, respectively, the quotient and residue of d_x when divided by $k-1$. Then the interpolation cost is bounded by*

$$\begin{aligned} \mathbb{C}_e(\epsilon) \leqslant & (m'+d_y+q_x+1)\left(\frac{1}{2}(m'+d_y+q_x)(k-1)+r_x\right) \\ & + m'+d_y+1. \end{aligned} \quad (7)$$

Proof: Let $x^{d_x}y^{d_y}$ be the last monomial of the polynomial $P(x,y)$ used in Theorem 7. The last monomial of $Q(x,y)$ is $x^{d_x}y^{m'+d_y}$, whose $(1,k-1)$-weighted degree is clearly $d_x + (k-1)(m'+d_y)$. Therefore substitution of $t = m'+d_y+q_x$, $u=r_x$ and $\ell = m'+d_y$ in (2) of Lemma 5 yields (7). □

Notes:

(1) Theorem 4 is a special case of Theorem 7 with $m'=m$ and $d_y=0$ (i.e. $P(x,y)$ univariate in x). In general, m' can be freely chosen to find the best bound on the interpolation cost $\mathbb{C}_e(\epsilon)$ for each error weight e.

(2) The more general bound of (7) is not given in closed form due to the inclusion of the intermediary parameters d_x, d_y. To obtain closed-form bounds (which are, in general, less tight), one can skip the calculation of d_x, d_y from the number of coefficients in $P(x,y)$, and instead seek upper bounds on $d_x + (k-1)(m'+d_y)$ and on d_y+m', which can be used as Δ and ℓ, respectively, in Lemma 5 to bound the interpolation cost of $Q(x,y)$. This process is described in the following. Let W be the number of coefficients in the polynomial $P(x,y)$. Starting from Δ, the $(1,k-1)$-weighted degree of $Q(x,y)$ we write

$$\begin{aligned} \Delta &\triangleq d_x + (k-1)(m'+d_y) \\ &= d_x + (k-1)d_y + (k-1)m' \\ &= d_{1,k-1}(P(x,y)) + (k-1)m' \\ &\leqslant \left\lfloor \sqrt{2(k-1)W} \right\rfloor - 1 + (k-1)m'. \end{aligned} \quad (8)$$

The last inequality follows from a straightforward inversion of (2). Similarly we obtain an upper bound on d_y+m'

$$\begin{aligned} \ell &\triangleq d_y + m' \\ &\leqslant \left\lfloor \sqrt{\left(\frac{k+1}{2(k-1)}\right)^2 + \frac{2(W-1)}{k-1}} - \frac{k+1}{2(k-1)} \right\rfloor + m'. \end{aligned} \quad (9)$$

The inequality follows from a similar inversion of (2), as detailed in Appendix A. Finally, we can substitute the upper bounds on Δ and ℓ from (8) and (9), respectively, in Lemma 5, with $W = e\binom{m+1}{2} + (n-e)\binom{m-m'+1}{2}$.

D. Interpolation costs for a sample RS code

In Sections III-B and III-C bounds were given for the error-weight dependent interpolation costs. Here we wish to explore the tightness of these bounds by using a standard interpolation algorithm (which gives a minimal interpolation polynomial for any error vector), and compare its interpolation costs to the bounds above. For that task, a GS decoder was implemented and run on an $[n,k] = [31,15]$ RS code. The interpolation multiplicity chosen for the decoder is $m=3$, which allows correcting 9 errors and has a worst case interpolation cost of $\mathbb{C}_{\text{wc}} = n\binom{m+1}{2} + 1 = 187$. The results are summarized in Table I below. Each row reflects a value of e and the columns compare observed results to the bounds. The columns tagged *observed* are the maximum, average and minimum interpolation costs used by the decoder. These numbers were generated using repeating runs ($\sim 10^5$ per e value) with random errors. For $e \leqslant 6$, no interpolation costs smaller than the closed form bound of Corollary 6 were observed. For $e=7$ the bound is attained in almost all instances, with few exceptions of up to a difference of 2. That is the case also for $e=8$, only that Corollary 8 is used to find an improved bound over Corollary 6. For $e=9$ the best upper bound for the interpolation cost is $\mathbb{C}_{\text{wc}}$. The results of this experimental study are that the upper bounds on interpolation costs provided here are tight in the worst case (max values attain the bounds for all e), and close to tight even in the average case. Hence, at least for this sample code, the bounds provide a succinct and reliable characterization of the decoder behavior. Going beyond this sample code to validate the upper bounds' tightness becomes a practical challenge for long codes with large interpolation multiplicities, and general analytical *lower* bounds seem hard to come by.

TABLE I
INTERPOLATION COSTS FOR THE $[31, 15]$ RS CODE WITH $m = 3$.

				observed		
#errors e	worst case $\mathbb{C}_{\rm wc}$	closed form (3)	improved (7)	max	average	min
0	187	88	-	88	88	88
1	187	100	-	100	100	100
2	187	112	-	112	112	112
3	187	124	-	124	124	124
4	187	136	-	136	136	136
5	187	149	-	149	149	149
6	187	164	-	164	164	164
7	187	179	-	179	178.95	177
8	187	194	183 , $m' = 1$	183	182.97	179
9	187	209	187 , $m' = 0$	187	186.93	184

IV. INTERPOLATION COST IN SOFT-DECISION DECODERS

The bounds derived in Section III apply to GS decoders that have a fixed interpolation multiplicity m. Under the fixed-multiplicity regime, each code coordinate obtains a single symbol hypothesis R_j from the channel, and each coordinate's hypothesis is equally interpolated into $Q(x, y)$. Having the interpolation multiplicity m specified as a decoder parameter allows us to derive bounds on the interpolation cost that depend on m (see (3), (7), (8) and (9)). One may wonder, and this section will try to answer, whether a similar technique can be used if the multiplicities are *not* specified a priori as a decoder parameter. This happens when soft decoding is carried out. When the channel is able to provide soft information on the code symbols, namely posterior probabilities of different hypotheses for each code coordinate, then the restriction of fixed multiplicity is lifted, and the individual multiplicities are calculated from the channel output. The regime of variable multiplicities is often referred to as *weighted* interpolation. Kötter and Vardy studied the problem of assigning multiplicity values to symbol hypotheses as a function of the posterior probabilities calculated from the channel output and statistics [12]. A block diagram of the Kötter-Vardy (KV) soft-decision decoding algorithm is provided in Figure 2. The input to the algorithm is a matrix of posterior probabilities whose elements are $\pi_{i,j} = \Pr(C_j = \beta_i | \mathrm{y}_j)$, where y_j is the channel observation at coordinate j. In words, $\pi_{i,j}$ is the probability that the symbol in coordinate j of the codeword is β_i, given the observed channel output for the coordinate. Given the $q \times n$ matrix $\Pi = [\pi_{i,j}]$, the KV algorithm computes a $q \times n$ interpolation multiplicity matrix $M = [m_{i,j}]$. In turn, the matrix M specifies the constraints to the interpolation algorithm: $D_{r,s}Q(\alpha_j, \beta_i) = 0$, for $\{(r, s) : r + s < m_{i,j}\}$. Equipped with the notation and basic ideas of weighted interpolation in the KV algorithm, we proceed to study the instantaneous-error dependent interpolation costs exhibited in soft decoders. Three parts comprise this study: first the bounds of Section III are extended to variable multiplicities in Section IV-A. Then in Section IV-B, for the limit of large interpolation costs, the channel statistics are used to bound interpolation multiplicities. Combining the results of the two sub-sections allows obtaining bounds on interpolation costs that only depend on the code and the channel, for arbitrary instantaneous channel outputs. Finally in Section IV-D, an experimental study of soft decoding of a sample RS code provides useful insights from a practical standpoint.

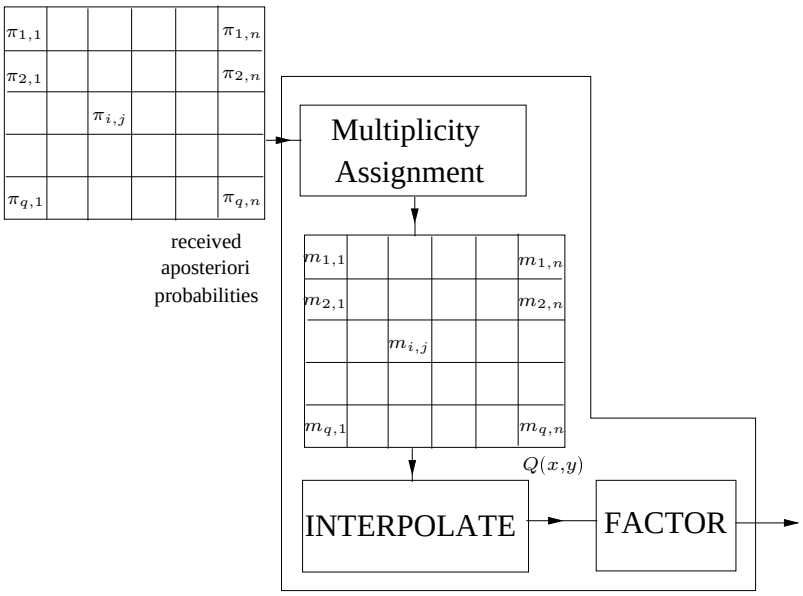

Figure 2. Block diagram of the Kötter-Vardy weighted list-decoding algorithm.

A. Bounds on weighted-interpolation costs

The complication in moving from hard-decision GS decoding to soft decision KV decoding stems from the following two impediments.

1) Multiplicities are not all identical
2) Multiplicities depend on the instantaneous noise

In the current sub-section, we ignore the second impediment, and focus on adapting the bounds to deal with the (milder) first one. For that we assume that we have an upper bound $\overline{m}_{i,j}$ on each of the interpolation multiplicities, i.e.

$$\forall (i, j),\ m_{i,j} \leqslant \overline{m}_{i,j}.$$

The next sub-section will be concerned with deriving such bounds, but for the time being we assume that $\overline{m}_{i,j}$ are given. As a reminder, our objective is to derive upper bounds on the interpolation cost, as a function of the number of *hard* errors caused by the channel[2]. Using the number of hard errors as the independent variable allows comparing the average running times of hard and soft decision decoders.

For notational convenience, we define the *ordered* multiplicity matrix $M_{\rm ord} = [\mu_{i,j}]$, whose column elements are ordered in non-increasing order: for a given channel instance, let $\mu_{i,j}$ be the multiplicity assigned to the symbol $\beta_{\chi_j(i)}$ on coordinate j, where for each j, the permutation $\chi_j(l)$ is defined over the elements $\{1, \ldots, q\}$, such that

$$\mu_{1,j} \geqslant \mu_{2,j} \geqslant \cdots \geqslant \mu_{q,j}.$$

[2]A channel output for a code coordinate is said to cause a hard error if a hard-decision decoder would use the wrong symbol hypothesis as input on that coordinate.

Note that according to this definition, $\mu_{1,j}$ is the largest multiplicity in column j. Given the multiplicity ordering in $M_{\rm ord}$, we define $\overline{\mu}_i$ to be an upper bound on any $i^{\rm th}$ multiplicity (satisfied for all coordinates j), so

$$\forall j,\ \mu_{i,j} \leqslant \overline{\mu}_i.$$

To link soft-decoding interpolation to the number of hard errors, we include a simple observation as the next proposition.

Proposition 9. *If there is no hard error in code coordinate j, then $\mu_{1,j}$ is assigned to the codeword symbol.*

Proof: By definition, the inputs to a hard-decision decoder are the symbols that maximize the posterior probability of the channel observations. Therefore, if there is no hard error on a coordinate, then the code symbol has the highest posterior probability. Having the highest posterior probability guarantees that the KV algorithm will assign to that symbol the largest multiplicity. This is true because in every step of the KV algorithm the multiplicity $m_{i,j}$ that has the largest value of $\pi_{i,j}/(m_{i,j}+1)$ is incremented. So clearly if $\pi_{i,j} \geqslant \pi_{i',j}$, then $m_{i,j} \geqslant m_{i',j}$. In particular, the symbol with the largest posterior probability has the largest interpolation multiplicity at every coordinate. □

We are now ready to provide an upper bound on interpolation costs for soft-decision decoding.

Theorem 10. *Let $\mathcal{E} = \{j_1, j_2, \ldots, j_e\}$ be the set of coordinates with hard errors. Then for any $m' \leqslant \overline{\mu}_1$, a polynomial of the form*

$$Q(x,y) = (y - f(x))^{m'} P(x,y)$$

is a weighted-interpolation polynomial, while $P(x,y)$ suffices to satisfy at most W interpolation constraints, where

$$\begin{aligned} W &= e \cdot \sum_{i=1}^{q} \binom{\overline{\mu}_i + 1}{2} \qquad (10) \\ &+ (n-e) \cdot \left[\binom{\overline{\mu}_1 - m' + 1}{2} + \sum_{i=2}^{q} \binom{\overline{\mu}_i + 1}{2} \right]. \end{aligned}$$

Proof: This is a simple generalization of Theorem 7. By Proposition 9, the $n-e$ coordinates with no hard errors have their largest multiplicity assigned to the code symbol $f(\alpha_j)$. Hence the polynomial $(y-f(x))^{m'}$ satisfies all the interpolation constraints up to order m', and $P(x,y)$ needs to satisfy only the remaining orders which are bounded by $\overline{\mu}_1 - m'$. □

A closed form bound now readily follows by substituting W above in (8) and (9), and substituting the resulting Δ and ℓ into Lemma 5. Note that W can be made equal to the expression for the hard-decision case (in Theorem 7) by setting $\overline{\mu}_1 = m$, and $\overline{\mu}_i = 0$ for $1 < i \leqslant q$.

B. Bounds on interpolation costs from channel statistics

In this sub-section, the task at hand is to find multiplicity bounds $\overline{\mu}_i$, which are used in the previous sub-section to bound interpolation costs. The method to obtain multiplicity bounds is by using information on the channel that governs the posterior probabilities input to the KV multiplicity-assignment algorithm. In general, this is a hard task since between the channel and the multiplicities $m_{i,j}$ lies the KV algorithm, to which small changes of inputs may change the output multiplicities in a way that is hard to analyze. Fortunately, however, in the limit of large interpolation costs, the behavior of the KV algorithm is conveniently characterizable using the following theorem from [12].

Theorem 11. ([12] Theorem 10) *As the sum of multiplicities $M \triangleq \sum_{i,j} m_{i,j}$ tends to infinity, the normalized multiplicity matrix converges to the normalized posterior-probability matrix*

$$\frac{1}{M} m_{i,j} \overset{M\to\infty}{\longrightarrow} \frac{1}{n} \pi_{i,j}.$$

Hence in the limit of large interpolation costs we can assume that $m_{i,j} \approx M\pi_{i,j}/n$. This property allows us to translate upper bounds on $\pi_{i,j}$ to upper bounds on $m_{i,j}$ by multiplying them by M/n, a ratio that does not depend on the instantaneous channel output. We now turn to seek upper bounds on posterior probabilities.

Let a code symbol x be defined over the alphabet of integers $\{1,\ldots,q\}$. Given a channel output y, the posterior probabilities $\pi_i(\mathrm{y})$ are

$$\pi_i(\mathrm{y}) = \Pr(\mathrm{x} = i | \mathrm{y}).$$

The length q *ordered* posterior-probability vector $\boldsymbol{\psi}(\mathrm{y}) = [\psi_1(\mathrm{y}), \ldots, \psi_q(\mathrm{y})]$ has all the elements $\pi_i(\mathrm{y})$, appearing in non-increasing order, i.e.

$$\boldsymbol{\psi}(\mathrm{y}) = \left[\pi_{i_1}(\mathrm{y}), \ldots, \pi_{i_q}(\mathrm{y})\right],$$

and $\pi_{i_1}(\mathrm{y}) \geqslant \pi_{i_2}(\mathrm{y}) \geqslant \cdots \geqslant \pi_{i_q}(\mathrm{y})$. Similarly to the notation for multiplicities, an upper bound on $\psi_i(\mathrm{y})$ (the $i^{\rm th}$ largest posterior probability), for any y, will be denoted $\overline{\psi}_i$ (while $\pi_i(\mathrm{y})$ and $\psi_i(\mathrm{y})$ are functions of the instantaneous channel output y, $\overline{\psi}_i$ is a value that only depends on the channel statistics). Our target now is to find, given the channel parameters, a vector of ordered upper bounds $\overline{\boldsymbol{\psi}} = \left[\overline{\psi}_1, \ldots, \overline{\psi}_q\right]$. If the channel is memoryless, then in the large-cost limit $\overline{\boldsymbol{\psi}}$ can be used to obtain upper bounds on interpolation multiplicities to be used in Theorem 10:

$$\forall j,\ \mu_{i,j} \leqslant \overline{\mu}_i \triangleq \frac{M}{n} \overline{\psi}_i.$$

Methods to compute vectors $\overline{\boldsymbol{\psi}}$ are now described as a sequence of propositions. We hereafter assume that the input alphabet to the channel are the integers $\{0,\ldots,q-1\}$. The channel output alphabet are real numbers in the interval $[0,q)$, and the channel is an additive-noise channel, taken modulo q:

$$\mathrm{y} = \mathrm{x} + \nu \pmod q,$$

where ν is a random variable that takes real values. The modulo operation can be interpreted as the channel wrapping around from $q-1$ back toward 0, as depicted in Figure 3. It is important to note that the modular definition is not necessary for obtaining the bounds, but it permits presenting more succinct bounds that depend only on the fractional part of the channel output y. If the channel does not wrap around, the calculations will have to be repeated for each integer part of y, and $\overline{\psi}_i$ will be taken as the maximum over all q possible

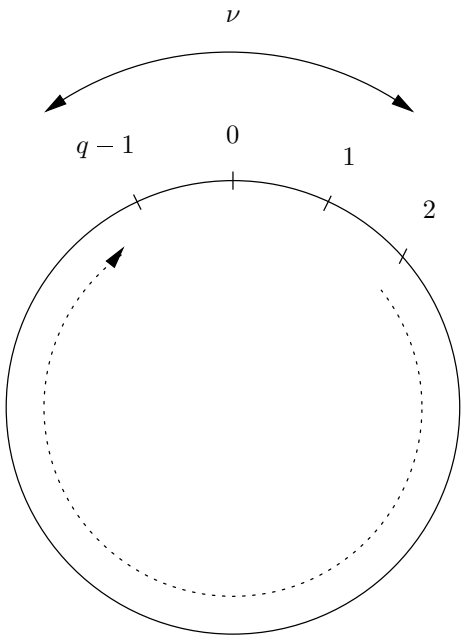

Figure 3. Channel wrap-around from $q-1$ back toward 0.

integer parts. Channel wrap around is motivated, for example, by multi-level cell solid-state storage, where limited-precision analog to digital (A/D) converters may cause wrap-around error when the noise is high enough. For any other channel exhibited in practical applications, wrap-around channels are essentially equivalent to non wrap-around ones when the noise distribution is narrow compared to the full dynamic range $[0, q)$. Therefore the analysis below is applicable to a broad spectrum of scenarios encountered in practice. Following a clean coding-theoretic presentation, in the sequel we interpret the channel outputs as direct evidence on the code symbols. However, it is an interesting future work to develop a similar analysis for important digital communications modulations like QAM and M-ary PSK.

Define $f(\mathrm{y}|\mathrm{x})$ to be the probability density function (pdf) of observing a real value y at the channel output, given the input symbol x. Also define $\Pr(\mathrm{x}|\mathrm{y})$ to be the probability that symbol x was transmitted, given an observed channel output y. Let $f_N(\nu)$ be the probability density function of the noise process, defined over the $(-\infty, \infty)$ interval of real numbers. Since $\mathrm{y} = \mathrm{x} + \nu \pmod q$, the two are related as

$$f(\mathrm{y}|\mathrm{x}) = f_N(\mathrm{y} \dot{-} \mathrm{x}),$$

where for any pair of elements a, b, the operator $a \dot{-} b$ is defined as

$$a \dot{-} b = \begin{cases} a - b & \text{if } |a-b| \leqslant q/2 \\ a - b - q \cdot \operatorname{sgn}(a-b) & \text{otherwise} \end{cases}$$

The operator $\dot{-}$ computes the signed distance between two points on the circle of Figure 3. Another way to look at the operator $\dot{-}$ is as standard subtraction taken modulo the interval $[-q/2, q/2)$. This latter view implies the associativity and commutativity of $\dot{-}$ to be used in later results.

In a series of propositions, we now provide tools for efficiently finding bounds on ψ_i. The more knowledge we have on the channel statistics, the easier is the task to find such bounds. The first fact about $\overline{\psi}_i$ is that they are completely characterizable by values of $\Pr(\mathrm{x}|\mathrm{y})$ in the interval $0 \leqslant \mathrm{y} < 1$.

Proposition 12. *If $\Pr(\mathrm{x}|\mathrm{y}) = \delta$ for some channel-input symbol x and channel output y, then there exists a channel input-output pair x_0 and y_0 such that $\Pr(\mathrm{x}_0|\mathrm{y}_0) = \delta$, and $0 \leqslant \mathrm{y}_0 < 1$.*

Proof: When all the inputs to the channel are equally probable (as is the case for a linear code with uniform distribution on the messages), the posterior probability is

$$\Pr(\mathrm{x}|\mathrm{y}) = \frac{f(\mathrm{y}|\mathrm{x})}{\sum_{l=0}^{q-1} f(\mathrm{y}|l)}.$$

From the additivity property of the noise

$$f(\mathrm{y}|\mathrm{x}) = f(\mathrm{y} \dot{-} 1|\mathrm{x} \dot{-} 1). \tag{11}$$

From the wrap-around property of the channel, subtracting one from y does not change the *values* of its modulo q distances from the alphabet symbols, thus

$$\sum_{l=0}^{q-1} f(\mathrm{y}|l) = \sum_{l=0}^{q-1} f(\mathrm{y} \dot{-} 1|l). \tag{12}$$

Combining (11) and (12) we get

$$\Pr(\mathrm{x}|\mathrm{y}) = \Pr(\mathrm{x} \dot{-} 1|\mathrm{y} \dot{-} 1).$$

So if $\Pr(\mathrm{x}|\mathrm{y}) = \delta$, then $\Pr(\mathrm{x} \dot{-} 1|\mathrm{y} \dot{-} 1) = \delta$ as well, and by induction it follows that $\Pr(\mathrm{x}_0|\mathrm{y}_0) = \delta$, with $0 \leqslant \mathrm{y}_0 < 1$. □

The next proposition shows that in the case of symmetric noise, it is sufficient to look for $\overline{\psi}_i$ in the interval $0 \leqslant \mathrm{y} < 0.5$.

Proposition 13. *If $f_N(\nu) = f_N(-\nu)$, and if $\Pr(\mathrm{x}|\mathrm{y}) = \delta$ for some channel-input symbol x and channel output y, then there exists a channel input-output pair x_1 and y_1 such that $\Pr(\mathrm{x}_1|\mathrm{y}_1) = \delta$, and $0 \leqslant \mathrm{y}_1 < 0.5$.*

Proof: From Proposition 12, $\Pr(\mathrm{x}_0|\mathrm{y}_0) = \delta$, and $0 \leqslant \mathrm{y}_0 < 1$. If $\mathrm{y}_0 < 0.5$ the proposition's statement is trivially proved, so assume $0.5 \leqslant \mathrm{y}_0 < 1$. Now

$$\begin{aligned} f(\mathrm{y}_0|\mathrm{x}_0) &= f_N(\mathrm{y}_0 \dot{-} \mathrm{x}_0) \\ &= f_N(\mathrm{x}_0 \dot{-} \mathrm{y}_0) \\ &= f_N((1 \dot{-} \mathrm{y}_0) \dot{-} (1 \dot{-} \mathrm{x}_0)) \\ &= f(1 - \mathrm{y}_0|1 \dot{-} \mathrm{x}_0). \end{aligned} \tag{13}$$

The transition to (13) follows by the symmetry of the noise pdf. A similar sequence of equalities can be applied to $\sum_{l=0}^{q-1} f(\mathrm{y}_0|l)$ in the denominator of the posterior probability, which together prove that

$$\Pr(1 \dot{-} \mathrm{x}_0|1 - \mathrm{y}_0) = \Pr(\mathrm{x}_0|\mathrm{y}_0) = \delta,$$

and clearly $0 \leqslant 1 - \mathrm{y}_0 < 0.5$. □

If the posterior probability $\Pr(\mathrm{x}|\mathrm{y})$ is monotone in y at the interval $0 \leqslant \mathrm{y} < 0.5$, either increasing or decreasing, then it is sufficient to look for bounds on ψ_i at the extremes of the search interval: $\mathrm{y} = 0$ and $\mathrm{y} = 0.5$. For x such that $\Pr(\mathrm{x}|\mathrm{y})$ is monotone *increasing* in y, the maximum in the interval $0 \leqslant \mathrm{y} < 0.5$ is attained at $\mathrm{y} = 0.5$. For x such that $\Pr(\mathrm{x}|\mathrm{y})$ is monotone *decreasing* in y, the maximum in the interval $0 \leqslant \mathrm{y} < 0.5$ is attained at $\mathrm{y} = 0$. One distribution for which

the posterior probability has the monotonicity property is the Normal distribution.

Proposition 14. *Let* $\mathrm{y} = \mathrm{x} + \nu$*, and* ν *be a random variable that is normally distributed with mean zero. If* $\mathrm{x} \div 1 \geqslant 0$ *then* $\Pr(\mathrm{x}|\mathrm{y})$ *is monotone increasing in* y *in the interval* $[0, 0.5)$*. If* $\mathrm{x} \div 1 < 0$ *then* $\Pr(\mathrm{x}|\mathrm{y})$ *is monotone decreasing in* y *in the interval* $[0, 0.5)$

Proof: See Appendix C. □

Example 1. *For* $q = 8$ *we want to find upper bounds on posterior probabilities under an additive channel that is normally distributed with* $\sigma = 2$*. From symmetry we can assume that* $0 \leqslant \mathrm{y} < 0.5$ *is the channel output. Then the bounds on the ordered posterior probabilities are as found in Table* II*. The channel inputs in the second column of the table appear in order of (cyclic) proximity to the interval* $[0, 0.5)$*. The* y *value in the third column, used to compute the bound, is* 0 *for* x *values with decreasing posterior probabilities (cases 2 and 3 in the proof of Proposition* 14*), and* 0.5 *for* x *values with increasing posterior probabilities (case 1 in the proof of Proposition* 14*). Note that summing up the probabilities in the last column gives* $1.09 > 1$*, due to their evaluation on different* y *values. This implies that a combination of posterior probabilities amounting to* $(\overline{\psi}_1, \ldots, \overline{\psi}_8)$ *cannot happen for any particular* y*. However, that compromise in the tightness of the bounds* $\overline{\psi}_i$ *is what allows them to be applied universally to arbitrary channel outputs* y*, and thus to provide bounds on interpolation costs that only use the code and channel parameters.*

TABLE II
UPPER BOUNDS ON THE POSTERIOR PROBABILITIES FOR A NORMALLY DISTRIBUTED CHANNEL NOISE WITH $\sigma = 2$.

Bound rank	Channel input x	y value for bound	Numerical value of $\Pr(\mathrm{x}|\mathrm{y})$
$\overline{\psi}_1$	$\mathrm{x} = 0$	$\mathrm{y} = 0$	$\Pr(0|0) = 0.210$
$\overline{\psi}_2$	$\mathrm{x} = 1$	$\mathrm{y} = 0.5$	$\Pr(1|0.5) = 0.202$
$\overline{\psi}_3$	$\mathrm{x} = 7$	$\mathrm{y} = 0$	$\Pr(7|0) = 0.185$
$\overline{\psi}_4$	$\mathrm{x} = 2$	$\mathrm{y} = 0.5$	$\Pr(2|0.5) = 0.157$
$\overline{\psi}_5$	$\mathrm{x} = 6$	$\mathrm{y} = 0$	$\Pr(6|0) = 0.127$
$\overline{\psi}_6$	$\mathrm{x} = 3$	$\mathrm{y} = 0.5$	$\Pr(3|0.5) = 0.095$
$\overline{\psi}_7$	$\mathrm{x} = 5$	$\mathrm{y} = 0$	$\Pr(5|0) = 0.068$
$\overline{\psi}_8$	$\mathrm{x} = 4$	$\mathrm{y} = 0.5$	$\Pr(4|0.5) = 0.045$

C. Direct calculation of the average interpolation cost

In the previous two sub-sections we derived bounds on the soft-decoding interpolation costs as a function of the instantaneous number of hard errors. The average interpolation cost can then be bounded by weighting these bounds with the probability distribution of the instantaneous-error counts. In this sub-section we pursue a *direct* calculation of the average interpolation cost, without first bounding the interpolation costs for individual instantaneous-error counts. Let $\lambda = M/n$ be a fixed decoder design parameter. For an interpolation polynomial of the form $(y - f(x))^{m'} P(x, y)$, the number of interpolation constraints that $P(x, y)$ has to satisfy (in the limit of large costs) is given by

$$\begin{aligned} W_{\mathrm{SD}} &= \frac{1}{2}\sum_{j \notin \mathcal{E}} (\lambda\pi_{1,j} - m')^2 + \frac{1}{2}\sum_{\substack{j \in \mathcal{E}, i \\ j \notin \mathcal{E}, i > 1}} (\lambda\pi_{i,j})^2 \\ &= \frac{1}{2}\sum_{j,i} (\lambda\pi_{i,j})^2 - m' \sum_{j \notin \mathcal{E}} (\lambda\pi_{1,j} - \frac{1}{2}m') \\ &= \frac{1}{2}\sum_{j,i} (\lambda\pi_{i,j})^2 - m' \sum_{j=1}^{n} \mathbb{1}[j \notin \mathcal{E}](\lambda\pi_{1,j} - \frac{1}{2}m'). \end{aligned}$$

The first equality rewrites the expression for W in (10) by taking the multiplicities as $\lambda\pi_{i,j}$ and approximating the binomial coefficients with square functions. The $\mathbb{1}[j \notin \mathcal{E}]$ is the indicator function that takes 1 if $j \notin \mathcal{E}$ and 0 otherwise. Taking the expectation and using the fact that the channel is memoryless we get

$$\begin{aligned} E[W_{\mathrm{SD}}] &= \\ n\sum_{i=1}^{q} \frac{\lambda^2}{2} E[\pi_i^2] - nm' &\left(\lambda E[\mathbb{1}[j \notin \mathcal{E}]\pi_1] - \frac{1}{2}m' E[\mathbb{1}[j \notin \mathcal{E}]]\right). \end{aligned} \tag{14}$$

$E[\mathbb{1}[j \notin \mathcal{E}]]$ equals $\Pr(j \notin \mathcal{E})$, the probability that there is no hard error in a received symbol. $E[\mathbb{1}[j \notin \mathcal{E}]\pi_1]$ equals $E[\pi_1 | j \notin \mathcal{E}] \Pr(j \notin \mathcal{E})$. Both follow from elementary probability theory. All of $\Pr(j \notin \mathcal{E})$, $E[\pi_1 | j \notin \mathcal{E}]$ and $E[\pi_i^2]$ can be computed from the channel statistics, hence the average interpolation cost can be calculated by evaluating (14) and adding the contribution of $(y - f(x))^{m'}$ to the cost.

D. Soft decoding of a sample RS code

To gain insight on weighted-interpolation costs in the non-asymptotic regime we veer to the experimental realm and simulate soft-decision decoding of a practical code. The same $[31, 15]$ RS code of Section III-D is used, but now over a channel whose description follows. The size $q = 32$ alphabet wraps around as in Figure 3 of sub-section IV-B. The noise samples added to the code symbols are drawn from an i.i.d. (independent and identically distributed) uniform random process with bounded support of $(-1, 1)$. This property implies that at most two symbol hypotheses per code location will be assigned non-zero interpolation multiplicities by the KV algorithm. A soft-decision decoder with at most two non-zero multiplicities is the closest possible to a hard-decision decoder (which assigns one non-zero interpolation multiplicity). We later show that even this small change introduces significant differences to the interpolation-cost behavior of the decoder. The soft-decision decoder we use has a worst-case interpolation cost identical to that of the hard-decision decoder of Section III-D: $\mathbb{C}_{\mathrm{wc}} = 187$. It is thus interesting to compare the instantaneous interpolation costs exhibited by the soft-decision decoder to those of the hard-decision decoder. In Figure 4, the interpolation cost is plotted as a function of the number of hard errors e caused by the channel. As discussed earlier in the section, the number e of hard errors is obtained by counting how many of the noise samples N_j have caused a crossover to an adjacent symbol. Mathematically: $e = |\{N_j : |N_j| > 0.5\}|$. Three of the four

plotted curves in Figure 4 are soft-decision (SD) results: the minimum (SD-Min), maximum (SD-Max), and average (SD-Avg) interpolation cost observed across multiple runs of the decoder for each number of hard errors introduced by the noise. The forth curve (HD-Avg) in Figure 4 is the average interpolation cost from a hard-decision (HD) decoder taken from Table I. Due to the low variability in HD interpolation cost seen in Table I, there is no need for similar HD-Min and HD-Max curves (since they would be very close to HD-Avg). We see that for low error weights the SD decoder requires on average higher interpolation costs compared to the HD decoder. For high error weights the average interpolation cost of SD drops slightly below that of HD, only due to a known property of the KV algorithm of failing to reach the designed interpolation cost [12, Sec. IV]. Figure 4 reveals another difference between SD and HD interpolation, in that SD exhibits significant variability around the average cost values. Both the relative flatness of SD-Avg and the high variability across runs indicate that in SD decoding, the dependence of the interpolation costs on the error weight is weaker, and new analysis techniques – as those given earlier in the section – are needed to predict the interpolation cost.

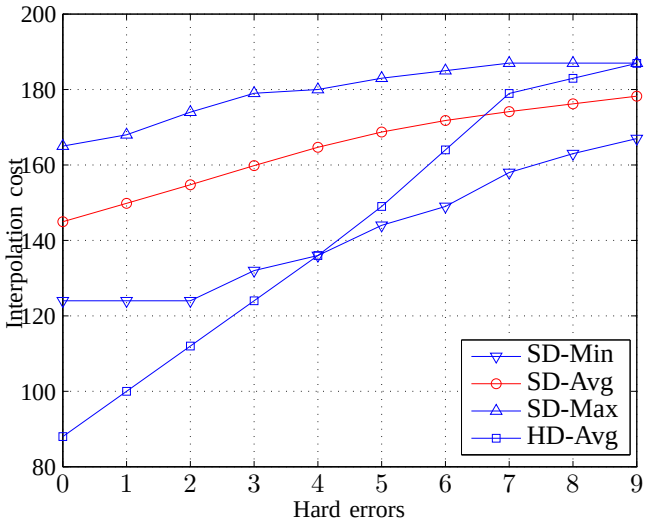

Figure 4. SD vs. HD interpolation cost as a function of the hard-error weight.

V. From Interpolation Cost to Decoding Complexity

In the preceding sections it has been proved that in many cases the interpolation cost is significantly lower than the worst case $\mathtt{C}_{\rm wc}$. That immediately means factorization algorithms would run faster in low-cost instances (due to short inputs provided from low-cost interpolations). However, the most computationally intensive part of the decoder is the interpolation algorithm. Unfortunately, a reduced interpolation cost does not automatically provide reduced running time of interpolation algorithms. Admittedly, we will see that state-of-the-art interpolation algorithms do not translate the savings in coefficients to savings in running time. That is true even in light of the fact that these algorithms do eventually output the lowest degree interpolation polynomials. This situation is unfortunate since the decoder fails to benefit from the worst-case/instantaneous-case gap that was pointed out earlier in the paper. We examine such behaviors of two interpolation algorithms in the case of reduced interpolation cost. We subsequently suggest modifications to the interpolation algorithms to improve their average-case running time.

A. Gaussian elimination interpolation

By formulating the interpolation problem as a system of homogeneous linear equations, Gaussian elimination stands out as a natural straightforward algorithm to solve it. This interpolation method is not the most efficient that exists, and we present it only to illustrate the connection between interpolation cost and running time. Gaussian elimination starts with a $(\mathtt{C}_{\rm wc}-1)\times\mathtt{C}_{\rm wc}$ matrix. The number of rows is the number of interpolation constraints and the number of columns is the worst case interpolation cost. Since that matrix is under-determined, upon termination we are guaranteed to reveal linearly dependent columns which result in coefficients of an interpolation polynomial. In Gaussian elimination, we successively add each column (multiplied by a scalar) to all the columns to its right. The more rightward the added column is, the more guaranteed zero elements it has when added to the columns to its right. To analyze the time complexity of the standard Gaussian elimination described above, we approximate the dimensions of the matrix by $\mathtt{C}_{\rm wc}\times\mathtt{C}_{\rm wc}$. The time complexity of a a $c\times c$ Gaussian elimination is known to approach $\frac{2}{3}c^3$ finite field operations (plus lower-order terms $o(c^3)$) [20, ch. IV]. This follows from

$$2\sum_{k=1}^{c}\sum_{j=k+1}^{c}(c-k+1)\rightarrow\frac{2}{3}c^3.$$

The outer sum runs on the columns being added; the inner sum runs on the $c-k$ columns to the right of each column k, and $c-k+1$ in the sum is the number of non-zero elements in the added column k, at the time of addition (all the elements above row k were already zeroed by previous steps of the algorithm). The factor 2 stands for two finite-field operations: multiplication by the leader/pivot ratio and addition. Thus using standard Gaussian elimination would consume $\frac{2}{3}\mathtt{C}_{\rm wc}^3$ (up to an additive constant) finite field operations, regardless of the actual interpolation cost of the decoding instance. The key to achieve better time complexity in cases of low interpolation cost is to *reorder* the operations of the algorithm to avoid unnecessary column operations. When a shorter interpolation polynomial exists, many columns on the right-hand side of the matrix are not needed for the interpolation. Therefore, one needs to avoid performing column operations on these columns. So rather then ordering the column operations ($a\rightarrow b$ denotes addition of a column a multiple to column b)

$$\{1\rightarrow 2,\ 1\rightarrow 3,\ldots,1\rightarrow c,\ 2\rightarrow 3,\ 2\rightarrow 4,\ \ \ldots,2\rightarrow c,\ldots\},$$

we follow the order

$$\{1\rightarrow 2,\ 1\rightarrow 3,\ 2\rightarrow 3,\ 1\rightarrow 4,\ 2\rightarrow 4,\ 3\rightarrow 4,\ 1\rightarrow 5,\ldots\}.$$

When the algorithm reaches an all-zero column, it stops. This algorithm is sometimes referred to as the Feng-Tzeng algorithm [6]. If we denote $c = \mathbb{C}_{\rm wc}$, $c' = \mathbb{C}_e(\epsilon)$ and $\gamma = \frac{c'}{c}$, then the time complexity of this reordered Gaussian elimination will be

$$2\sum_{k=1}^{c'}\sum_{j=k+1}^{c}(c'-k+1) = 2\sum_{k=1}^{c'}(c-k)(c'-k+1)$$

$$= 2\sum_{k=1}^{c'} cc' - k(c+c') + k^2 - k + c.$$

Now taking only the first three (quadratic) terms in the sum (the other two are negligible for large c):

$$2\sum_{k=1}^{c'} cc' - k(c+c') + k^2 \to 2\left[cc'^2 - \frac{c'^2(c+c')}{2} + \frac{c'^3}{3}\right],$$

where the $\to$ operator neglects $o(c^3)$ terms. The last expression becomes

$$cc'^2 - \frac{1}{3}c'^3 = \frac{2}{3}\mathbb{C}_{\rm wc}^3\left(\frac{3}{2}\gamma^2 - \frac{1}{2}\gamma^3\right),$$

and that yields a $\frac{3}{2}\gamma^2 - \frac{1}{2}\gamma^3$ factor of saving.

B. The standard interpolation algorithm

Now we wish, for the same analysis we previously applied to Gaussian elimination, to consider the standard, most efficient interpolation algorithm used in RS list decoding. This algorithm and its variants were intensively studied in the literature, e.g., in [10], [15], [13], [1]. We present a somewhat rough sketch of the algorithm here, in order to focus on the computational issue at hand. The key idea of the algorithm [14], [11], is to interpolate $L+1$ polynomials, each with a different y-degree, where L is an upper bound on the y-degree, calculated from the worst-case interpolation cost, and upon termination select the one with the lowest interpolation cost. By fixing the y-degrees throughout the update process, a "greedy" update rule cumulatively satisfies all interpolation constraints, and is guaranteed to output minimal polynomials for each y-degree. In the main loop of the algorithm, the lowest-degree candidate polynomial (out of the $L+1$ y-degree candidates) is used to update the rest of the candidate polynomials such that they will meet the current interpolation constraint. The algorithm is given as pseudo-code in Algorithm 1. The loop index l runs on the $\mathbb{C}_{\rm wc}-1$ interpolation constraints. Each constraint l represents a triple r, s, i, where r, s is the order of the Hasse derivative, and i is the codeword coordinate. A discrepancy $\delta_j^{(l)}$ of a polynomial Q_j with respect to an interpolation constraint l is the non-zero value to which the r, s Hasse derivative evaluates at point (α_i, R_i), i.e.

$$\delta_j^{(l)} = D_{r,s}Q_j(\alpha_i, R_i) \neq 0.$$

The *argmin* operator in Algorithm 1 selects the index j of the polynomial Q_j whose highest monomial has the lowest index with respect to the standard monomial ordering. In line *(2)* of the Algorithm 1 pseudo code, the non-zero discrepancy of that lowest degree polynomial is used to eliminate the discrepancies of other, higher degree, polynomials, without changing their degrees. Next the lowest-degree polynomial is updated (with degree increase) as to satisfy constraint l as well.

Algorithm 1: Standard interpolation algorithm (sketch)

Initialize
$Q_j := y^j,\ \forall j \in \{0, \ldots, L\}$ `// L is a bound on the` y-`degree of the output polynomial`
for $l := 1$ **to** $\mathbb{C}_{\rm wc} - 1$ `// interpolation constraints`
(1) $\delta_j^{(l)} :=$ discrepancy of Q_j with respect to constraint l
$j^\star := \text{argmin}(j : \delta_j^{(l)} \neq 0)$
forall j with non-zero $\delta_j^{(l)}$
(2) for $j \neq j^\star$ update Q_j with no change in degree
for $j^\star$ update $Q_{j^\star}$ with degree increment
output Q_j with minimal degree

Analysis of the complexity of the algorithm: it iterates on $\mathbb{C}_{\rm wc} - 1$ constraints and in each iteration performs $L+1$ polynomial evaluations in *(1)* and (at most) $L+1$ polynomial updates in *(2)*. The running time of both the evaluations and updates is linearly proportional to the number of monomials with non-zero coefficients, which in the worst case equals $\mathbb{C}_{\rm wc}$. Thus the total running time of the algorithm is

$$T_{\rm wc} = (\mathbb{C}_{\rm wc}-1)(L+1)\mathbb{C}_{\rm wc}(t_{\rm em}+t_{\rm um}) \approx (L+1)\mathbb{C}_{\rm wc}^2(t_{\rm em}+t_{\rm um}), \quad (15)$$

where $t_{\rm em}$ and $t_{\rm um}$ are the times for evaluation and update *per monomial*, respectively.

We next observe that the running time will not be significantly better in cases when the final interpolation cost is small. The reason being that the computation load is dominated by operations on non-minimal polynomials. Even if a polynomial $Q_{j^\star}$ satisfies all interpolation constraints with low cost, the algorithm does not know the identity of that $j^\star$ in advance, and has to successively update all polynomials Q_j that have higher costs. It also does not a priori know the final required cost and thus cannot exclude polynomials with higher costs during computation. Consequently, this fast interpolation algorithm will have an average-case running time not better than that of the worst case, even if most of the decoding instances are low-error ones.

To fix that undesirable behavior, we need to modify the algorithm as to reorder its operations, as was done in the previous sub-section for Gaussian elimination, but in a less obvious way. The operation ordering will be in a way that discrepancy calculations and polynomial updates are performed only on polynomials whose coefficient counts are *guaranteed* to be at most the final interpolation cost. It turns out that this can be done by modifying the algorithm iteration, with no increase in worst-case time complexity (but at a cost of requiring additional storage to hold older versions of polynomials Q_j that are needed in future updates). The idea is that as long as a polynomial is not the lowest-degree candidate, its updates are postponed until it does become the lowest-degree one (as a result of updates to the other polynomials). When that happens, we perform all postponed updates on this polynomial using previously stored polynomials and discrepancies. That way it is able to "catch up" and satisfy all past interpolation

constraints. In Algorithm 2 the modified algorithm is presented as pseudo-code.

Algorithm 2: Interpolation with improved average running time

Initialize
$Q_j := y^j,\ \forall j \in \{0, \ldots, L\}$
$l_j := 0,\ \forall j \in \{0, \ldots, L\}$ `// constraint pointer for each` j
$j^\star := 0$ `//` $j^\star = \mathrm{argmin}_j Q_j$
while $l_{j^\star} < \mathbb{C}_{\mathrm{wc}} - 1$ `// while no` Q_j `satisfies all constraints`
  $l_{j^\star}++$
  calc $\delta_{j^\star}^{(l_{j^\star})}$
  if $\delta_{j^\star}^{(l_{j^\star})} = 0$ **continue**
(*) **read** $(\Delta[j, l_{j^\star}]\ ,\ \mathcal{Q}[j, l_{j^\star}])$ `// read poly with discrepancy`$\neq 0$
  if (exists) **update** $Q_{j^\star}$ with no change in degree
  else
    store $(\Delta[j^\star, l_{j^\star}]\ ,\ \mathcal{Q}[j^\star, l_{j^\star}]) \leftarrow (\delta_{j^\star}^{(l_{j^\star})}\ ,\ Q_{j^\star})$
    update $Q_{j^\star}$ with degree increment
    $j^\star := \mathrm{argmin}_j Q_j$ `// proceed with the best candidate`
output $Q_{j^\star}$

The index $j^\star$ is the y-degree of the current lowest-degree candidate, and it is initialized to 0. For each y-degree j, the index l_j designates that the polynomial Q_j satisfies all the interpolation constraints $\{1, 2, \ldots, l_j\}$. This is is contrast to Algorithm 1, where there is only one global counter l for the constraints, and all polynomials satisfy the same cumulative kernel $\{1, 2, \ldots, l\}$. *read* in *(*)* refers to the action of reading $\mathcal{Q}[j, l_{j^\star}]$ (if exists), a stored polynomial Q_j that had a non-zero discrepancy on $l_{j^\star}$, and its corresponding discrepancy value $\Delta[j, l_{j^\star}]$. For each index l, there is at most one stored polynomial $\mathcal{Q}[j, l]$ (the first Q_j that had a non-zero discrepancy on l). Hence reading the stored polynomial can be done trivially in constant time (accessing a fixed index l), assuming space is allocated for up to $\mathbb{C}_{\mathrm{wc}}$ polynomials.

The total amount of memory required by Algorithm 2 to store past interpolation polynomials is upper bounded by $\mathbb{C}_{\mathrm{wc}}^2 \log_2(q)$ bits. That is because each interpolation constraint $l < \mathbb{C}_{\mathrm{wc}}$ has at most one stored polynomial with at most $\mathbb{C}_{\mathrm{wc}}$ q-ary coefficients. In fact, the actual memory requirement will be (predictably) lower by a constant factor, since for low l indices the maximum number of coefficients in the stored polynomial is bounded strictly below $\mathbb{C}_{\mathrm{wc}}$.

The following facts facilitate the verification of the correctness of the algorithm and its complexity.

(1) *Claim:* Discrepancy calculations and polynomial updates are performed only on polynomials with degrees lower than or equal to the final interpolation polynomial. *Justification:* These operations are performed only on the $j^\star$ polynomials, which are the lowest degree of all ($j^\star := \mathrm{argmin}_j Q_j$), and in particular cannot have higher degrees than the final polynomial.

(2) *Claim:* When a new polynomial is selected in $j^\star := \mathrm{argmin}_j Q_j$, it satisfies all the constraints $\{1, 2, \ldots, l_{j^\star}\}$. *Justification:* The only place where $l_{j^\star}$ is incremented is in a loop iteration at the end of which $Q_{j^\star}$ is updated to satisfy constraint $l_{j^\star}$. Follows by induction on $l_{j^\star}$.

(3) *Claim:* The minimal polynomial is the first to achieve $l_{j^\star} = \mathbb{C}_{\mathrm{wc}} - 1$. *Justification:* Follows from the invariants of the previous claims: only minimal polynomials are updated (1), at the end of the update all constraints up to $l_{j^\star}$ are satisfied (2).

To summarize, claim (1) establishes the *complexity* of the algorithm, and claims (2) and (3) prove its correctness (the output polynomial satisfies all the constraints, and is minimal). As a result, the time complexity of this modified algorithm is

$$T_e \quad = \quad (\mathbb{C}_{\mathrm{wc}} - 1)(L' + 1)\gamma \mathbb{C}_{\mathrm{wc}}(t_{\mathrm{em}} + t_{\mathrm{um}}).$$

The first term on the right-hand side of the above equation is the number of interpolation constraints, the second, $L' + 1 \leqslant L + 1$, is the number of y-degree polynomials that have initial interpolation costs below $\gamma \mathbb{C}_{\mathrm{wc}}$. The next term $\gamma \mathbb{C}_{\mathrm{wc}}$ is an upper bound on the number of monomials in each of the $L' + 1$ polynomials. This gives

$$T_e \quad \approx \quad \gamma(L' + 1)\mathbb{C}_{\mathrm{wc}}^2(t_{\mathrm{em}} + t_{\mathrm{um}}). \tag{16}$$

Note that the $L - L'$ polynomials with higher initial costs are never updated by the algorithm, since they cannot be selected as leaders $j^\star$. Comparing (16) to (15), the improved algorithm is faster by a factor

$$\frac{1}{\gamma}\frac{L+1}{L'+1}. \tag{17}$$

The value of this savings factor depends on the code, decoder parameters and error weight. A quantitative evaluation of the complexity savings factor is conducted in the next sub-section.

C. Quantitative evaluation of the complexity savings

To evaluate the significance of the complexity savings offered by Algorithm 2 in real decoders, we compare the total number of arithmetic operations used during interpolation with the standard decoder of Algorithm 1. The operations we count in the comparison are additions, multiplications and exponentiations over the code-alphabet finite field. Assuming that the complexity of each operation does not depend on its arguments, these counts completely characterize the complexity of the interpolation algorithms. For the $[31, 15]$ RS code and multiplicity $m = 3$ hard-decision decoder, the average operation counts are given in Table III as a function of the number of hard errors. The numbers in the table are in units of one thousand operations, and were obtained by 500 repeated runs of the algorithms with randomly selected errors (the average counts became stable well before 500 runs, so more runs would not have improved the estimates). From left to right, the column pairs of Table III list the number of additions, multiplications, exponentiations and total operations. Looking at the rightmost column pair of Table III, it is observed that the new decoder with Algorithm 2 improves over the known state-of-the-art for every number of instantaneous errors, including those cases when decoding is beyond the unique-decoding bound (9 errors in this example). The improvement is especially significant for low numbers of instantaneous errors, with factor 2 or higher reduction in complexity for 4 or less

TABLE III
AVERAGE INTERPOLATION OPERATION COUNT COMPARISON FOR THE $[31, 15]$ RS CODE WITH $m = 3$. NUMBERS GIVEN IN UNITS OF 10^3 OPERATIONS.

#errors	Av. adds		Av. muls		Av. exps		Av. total	
	Alg.1	Alg.2	Alg.1	Alg.2	Alg.1	Alg.2	Alg.1	Alg.2
1	108	41	195	74	172	66	475	181
2	113	46	203	85	177	75	493	206
3	118	53	210	96	183	85	511	234
4	124	61	219	110	189	96	532	267
5	130	98	228	174	194	150	552	422
6	136	114	235	201	197	171	568	486
7	141	131	243	231	200	196	584	558
8	146	136	249	238	203	200	598	574
9	150	142	254	246	204	204	608	592

errors. The reason the complexity saving factors are more significant than the interpolation cost reduction factors (in Table I) is that Algorithm 2 does not update all the $L + 1$ candidate polynomials when the number of errors is small. The implication of these results is that the average decoding time is significantly reduced with no effect on the worst-case decoding time. This is in contrast to other improvements of the average-case complexity, e.g. [19], which adversely affect the worst-case complexity in cases of large numbers of errors. The cost of this improvement is in memory used to store previous versions of interpolation polynomials; for the current example the required amount of memory is 10KB, not a significant amount in a typical high-rate decoder implementation, which uses large data buffers to process multiple received words in a pipeline.

To complete the quantitative evaluation of the improved algorithm, we detail the operation counts for a more realistic code for practical use. The $[31, 15]$ RS code used so far in the paper served as a convenient example to understand the behavior of interpolation costs. Now we wish to consider the $[127, 60]$ RS code over GF(128) as a sample code closer to codes used in practice. The reason we again choose the rate to be around $1/2$ is to obtain good list decoding capabilities that will give meaningful results. Another motivation to use a code that can correct many errors is that such codes are more prone to large variations in the number of instantaneous errors, and thus can enjoy greater savings in average-case running times. For the $[127, 60]$ RS code we choose a list decoder with multiplicity parameter $m = 3$. The number of correctable errors of this decoder is 36, higher by 3 than that of a traditional unique decoder, and lower by 4 than the maximal-radius GS decoder, which requires $m = 31$ (factor 10 higher m would cost about factor 10^4 higher complexity). The operation counts of the decoder are plotted in Figure 5.

For each number of instantaneous errors between 0 and 36, the operation counts (additions+multiplications+exponentiations) are plotted in units of 10^6 operations. The upper curve (square markers) shows the counts for Algorithm 1, and the lower curve (x markers) shows the counts for Algorithm 2. Each point was obtained from 20 runs with the same parameters (and randomly chosen error vectors), which is sufficient to smooth the relatively small variations.

The most apparent fact from Figure 5 is that the complexity

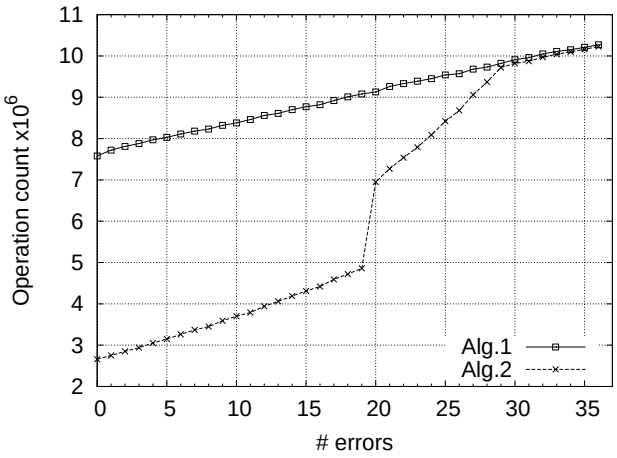

Figure 5. Operation counts for Alg.1 (upper) and Alg.2 (lower) for the $[127, 60]$ RS code with $m = 3$. Counts in units of 10^6 operations plotted for each instantaneous number of errors between 0 and 36.

reduction of Algorithm 2 is significant – close to factor 3 faster at the low range of errors, and close to factor 2 at the medium range. With the multiplier of millions of operations, this translates to a clear practical advantage. Unlike the standard decoder that follows a roughly constant slope, the improved decoder exhibits multiple different slopes in the interpolation-complexity curve. In each (approximately) linear segment of the Alg.2 curve, the worst-case y-degree (denoted L') of the interpolation polynomial is different. Since the complexity of Algorithm 2 has a linear dependence in L' (because $L' + 1$ is the number of simultaneous interpolation polynomials being updated, see (17)), different L' values result in different complexity slopes. It is important to re-emphasize that the lower slopes due to small L' are obtained "automatically," without speculation by Algorithm 2 on the number of instantaneous errors. As a result, from the right part of Figure 5 it can be observed that the complexity improvements are achieved with no sacrifice to the worst-case running time. This is an important advantage of Algorithm 2 over decoders, such as [19], that trade average-case complexity with worst-case complexity by using successions of lower-radius decoders. The amount of memory required to store past polynomials is 250KB in this example, clearly not a prohibitive requirement.

VI. CONCLUSION

The results presented in this paper provide a general framework to improve, and quantify the improvement, of the average complexity of Reed-Solomon code algebraic list decoders. However, each of the algorithmic/analytical tools described here can be further studied and extended. The improved interpolation algorithm can be further optimized for time and space complexities. More refined channel models, e.g. additive noise channels over PSK and QAM signal constellations can be considered for bounding the resulting posterior probabilities and in turn the average interpolation costs. In addition, extending the analytic and algorithmic results from Reed-Solomon

codes to algebraic-geometry codes is an interesting yet still open area.

VII. Acknowledgement

The authors would like to thank the associate editor and anonymous reviewers of the paper for their comments and suggestions that greatly improved the presentation. In particular, they proposed the closed form expression of the cost ratio in (4), and suggested the direct average-case analysis we pursued in Section IV-C, as well as the quantitative evaluation of the operation counts that closes Section V.

Appendix A
Inversion of (2)

Equation (2) states that

$$J = (t+1)\left(\frac{1}{2}t(k-1)+u\right)+l+1.$$

We first substitute $J = W$, the number of coefficients in $P(x,y)$, and $l = d_y$, by definition. Then we observe the inequalities $u \geqslant 0$ and $t \geqslant d_y$, the latter of which is because $(k-1)d_y$ is a lower bound on $d_{1,k-1}(P(x,y))$. Altogether we get

$$W \geqslant (d_y+1)\left[\frac{1}{2}d_y(k-1)\right]+d_y+1.$$

Solving the quadratic inequality for d_y we get

$$d_y \leqslant \sqrt{\left(\frac{k+1}{2(k-1)}\right)^2+\frac{2(W-1)}{k-1}}-\frac{k+1}{2(k-1)}.$$

Appendix B
The no-error case

Theorem 15. *Let $\rho = k/n$ be the rate of the RS code. When the received word R is a codeword, the interpolation cost $\mathtt{C}_0$ satisfies*

$$\mathtt{C}_0 \leqslant \lceil \rho \mathtt{C}_{\mathrm{wc}} \rceil.$$

Proof: When R is a codeword, R can be interpolated by the bivariate polynomial $Q(x,y) = (y-f(x))^m$. The last monomial of $(y-f(x))^m$ in the monomial order is y^m, whose $(1,k-1)$-degree is $m(k-1)$ and $(0,1)$-degree is m. The corresponding parameters of Lemma 5 are $\Delta = m(k-1)$ and $\ell = m$. Thus to get $\mathtt{C}$, the worst-case number of coefficients of the polynomial $Q(x,y)$, we substitute $t = m$, $u = 0$ and $\ell = m$ in (2)

$$\begin{aligned}\mathtt{C} &= (m+1)\frac{1}{2}m(k-1)+m+1\\ &= (k-1)\binom{m+1}{2}+m+1\\ &= k\binom{m+1}{2}-\frac{1}{2}(m+1)(m-2).\end{aligned}$$

Now substituting $k\binom{m+1}{2} = \rho\mathtt{C}_{\mathrm{wc}}-\rho$ we get

$$\mathtt{C} = \rho\mathtt{C}_{\mathrm{wc}}-\rho-\frac{1}{2}(m+1)(m-2).$$

It is not hard to see that $\frac{1}{2}(m+1)(m-2) \geqslant -1$, and so

$$\mathtt{C} \leqslant \rho\mathtt{C}_{\mathrm{wc}}-\rho+1 = \lceil \rho\mathtt{C}_{\mathrm{wc}} \rceil.$$

The last equality follows because $\rho\mathtt{C}_{\mathrm{wc}}-\rho+1$ is an integer. Since $\mathtt{C}_0 \leqslant \mathtt{C}$ the theorem follows. □

Appendix C
Proof of Proposition 14

Proof: The monotonicity of the *likelihood* function $f(\mathrm{y}|\mathrm{x})$ is easily obtained from the monotonicity of $f_N(\mathrm{y} \dot{-} \mathrm{x})$ for the normal distribution's pdf (monotone increasing for negative ν and monotone decreasing for positive ν). The objective now is to show that monotonicity still applies when the posterior probability is considered. For ease of notation, we denote the pdf of the normal distribution as

$$f_N(\nu) = \frac{1}{\sigma}\phi\left(\frac{\nu}{\sigma}\right),$$

where $\phi(x) = 1/\sqrt{2\pi}\exp(-x^2/2)$ and σ^2 is the distribution's variance. We start by substituting the normal pdf into the posterior probability

$$\Pr(\mathrm{x}|\mathrm{y}) = \frac{f(\mathrm{y}|\mathrm{x})}{\sum_{l=0}^{q-1} f(\mathrm{y}|l)} = \frac{\phi\left(\frac{\mathrm{y}\dot{-}\mathrm{x}}{\sigma}\right)}{\sum_{l=0}^{q-1}\phi\left(\frac{\mathrm{y}\dot{-}l}{\sigma}\right)} \triangleq \frac{g(\mathrm{y})}{h(\mathrm{y})}.$$

Using the quotient rule, we write the numerator of the derivative of $\Pr(\mathrm{x}|\mathrm{y})$ with respect to y, ignoring positive multiplicative constants (those can be passed to the denominator without affecting the sign of the derivative):

$$g'(\mathrm{y})h(\mathrm{y})-g(\mathrm{y})h'(\mathrm{y})$$

$$\begin{aligned}&= \phi\left(\frac{\mathrm{y}\dot{-}\mathrm{x}}{\sigma}\right)\sum_{l=0}^{q-1}(\mathrm{y}\dot{-}l)\phi\left(\frac{\mathrm{y}\dot{-}l}{\sigma}\right)\\ &\quad -(\mathrm{y}\dot{-}\mathrm{x})\phi\left(\frac{\mathrm{y}\dot{-}\mathrm{x}}{\sigma}\right)\sum_{l=0}^{q-1}\phi\left(\frac{\mathrm{y}\dot{-}l}{\sigma}\right)\\ &= \phi\left(\frac{\mathrm{y}\dot{-}\mathrm{x}}{\sigma}\right)\sum_{l=0}^{q-1}((\mathrm{y}\dot{-}l)-(\mathrm{y}\dot{-}\mathrm{x}))\phi\left(\frac{\mathrm{y}\dot{-}l}{\sigma}\right).\end{aligned}$$

We hereafter assume that q is even. The case of odd q is substantially similar and is omitted.
Case 1: $1 \leqslant \mathrm{x} \leqslant q/2$.
First we split the sum into two sums to remove the $\dot{-}$ operators. Since $0 \leqslant \mathrm{y} < 0.5$, $\mathrm{y}\dot{-}l = \mathrm{y}-l$ for $0 \leqslant l \leqslant q/2$, and $\mathrm{y}\dot{-}l = \mathrm{y}-l+q$ for $q/2+1 \leqslant l \leqslant q-1$,

$$\sum_{l=0}^{q-1}((\mathrm{y}\dot{-}l)-(\mathrm{y}\dot{-}\mathrm{x}))\phi\left(\frac{\mathrm{y}\dot{-}l}{\sigma}\right) \tag{18}$$

$$
\begin{aligned}
&= \sum_{l=0}^{q/2}((\mathrm{y}-l)-(\mathrm{y}-\mathrm{x}))\phi\left(\frac{\mathrm{y}-l}{\sigma}\right) \\
&\quad + \sum_{l=q/2+1}^{q-1}((\mathrm{y}-l+q)-(\mathrm{y}-\mathrm{x}))\phi\left(\frac{\mathrm{y}-l+q}{\sigma}\right) \\
&= \sum_{l=0}^{q/2}(\mathrm{x}-l)\phi\left(\frac{\mathrm{y}-l}{\sigma}\right) \qquad (19) \\
&\quad + \sum_{l=q/2+1}^{q-1}(\mathrm{x}+q-l)\phi\left(\frac{\mathrm{y}-l+q}{\sigma}\right). \\
&\geqslant \sum_{l=\mathrm{x}+1}^{q/2}(\mathrm{x}-l)\phi\left(\frac{\mathrm{y}-l}{\sigma}\right) \qquad (20) \\
&\quad + \sum_{l=q/2+1}^{q-\mathrm{x}}(\mathrm{x}+q-l)\phi\left(\frac{\mathrm{y}-l+q}{\sigma}\right).
\end{aligned}
$$

Where (19) is obtained by simple rearrangement, and (20) follows from discounting non-negative terms in the sum. Next, manipulating the indices of the second sum of (20), and rewriting as a single sum we get

$$
\begin{aligned}
&= \sum_{l=\mathrm{x}+1}^{q/2}\left[(\mathrm{x}-l)\phi\left(\frac{\mathrm{y}-l}{\sigma}\right)\right. \\
&\quad \left. + (\mathrm{x}+l-1)\phi\left(\frac{l-(1-\mathrm{y})}{\sigma}\right)\right].
\end{aligned}
$$

Finally, examining the two terms in the sum, for x in the specified range, $\mathrm{x}+l-1$ is positive and is larger in absolute value than the negative $\mathrm{x}-l$. In addition, $1-\mathrm{y}>\mathrm{y}$ implies that the argument $(\mathrm{y}-l)/\sigma$ is larger in absolute value than $(l-(1-\mathrm{y}))/\sigma$. These two facts combine to prove that all the summands in the last sum are positive, and that the posterior probability is monotone increasing.

Case 2: $q/2+1 \leqslant \mathrm{x} \leqslant q-1$.

Splitting the sum of (18) into two sums now yields

$$
\begin{aligned}
&\sum_{l=0}^{q-1}((\mathrm{y}\dot{-}l)-(\mathrm{y}\dot{-}\mathrm{x}))\phi\left(\frac{\mathrm{y}\dot{-}l}{\sigma}\right) \\
&= \sum_{l=0}^{q/2}((\mathrm{y}-l)-(\mathrm{y}-\mathrm{x}+q))\phi\left(\frac{\mathrm{y}-l}{\sigma}\right) \\
&\quad + \sum_{l=q/2+1}^{q-1}((\mathrm{y}-l+q)-(\mathrm{y}-\mathrm{x}+q))\phi\left(\frac{\mathrm{y}-l+q}{\sigma}\right). \\
&= \sum_{l=0}^{q/2}(\mathrm{x}-l-q)\phi\left(\frac{\mathrm{y}-l}{\sigma}\right) \qquad (21) \\
&\quad + \sum_{l=q/2+1}^{q-1}(\mathrm{x}-l)\phi\left(\frac{\mathrm{y}-l+q}{\sigma}\right) \\
&\leqslant \sum_{l=q-\mathrm{x}+1}^{q/2}(\mathrm{x}-l-q)\phi\left(\frac{\mathrm{y}-l}{\sigma}\right) \qquad (22) \\
&\quad + \sum_{l=q/2+1}^{\mathrm{x}}(\mathrm{x}-l)\phi\left(\frac{\mathrm{y}-l+q}{\sigma}\right).
\end{aligned}
$$

Where (21) is obtained by simple rearrangement, and (22) follows from discounting non-positive terms in the sum. Next, manipulating the indices of the second sum of (22), and rewriting as a single sum we get

$$
\begin{aligned}
&= \sum_{l=0}^{\mathrm{x}-q/2-1}\left[(\mathrm{x}-3q/2+l)\phi\left(\frac{\mathrm{y}+l-q/2}{\sigma}\right)\right. \\
&\quad \left. + (\mathrm{x}-q/2-1-l)\phi\left(\frac{q/2-l-(1-\mathrm{y})}{\sigma}\right)\right].
\end{aligned}
$$

Finally, examining the two terms in the sum, for x in the specified range, $\mathrm{x}-3q/2+l$ is negative and is larger in absolute value than the positive $\mathrm{x}-q/2-1-l$. In addition, $1-\mathrm{y}>\mathrm{y}$ implies that the argument $(q/2-l-(1-\mathrm{y}))/\sigma$ is larger in absolute value than $(\mathrm{y}+l-q/2)/\sigma$. These two facts combine to prove that all the summands in the last sum are negative, and that the posterior probability is monotone decreasing.

Case 3: $\mathrm{x}=0$.

Splitting the sum of (18) into two sums now yields

$$
\begin{aligned}
&\sum_{l=0}^{q-1}((\mathrm{y}\dot{-}l)-(\mathrm{y}\dot{-}\mathrm{x}))\phi\left(\frac{\mathrm{y}\dot{-}l}{\sigma}\right) \\
&= \sum_{l=0}^{q/2}((\mathrm{y}-l)-\mathrm{y})\phi\left(\frac{\mathrm{y}-l}{\sigma}\right) \\
&\quad + \sum_{l=q/2+1}^{q-1}((\mathrm{y}-l+q)-\mathrm{y}))\phi\left(\frac{\mathrm{y}-l+q}{\sigma}\right) \\
&= \sum_{l=0}^{q/2}(-l)\phi\left(\frac{\mathrm{y}-l}{\sigma}\right) \qquad (23) \\
&\quad + \sum_{l=q/2+1}^{q-1}(q-l)\phi\left(\frac{\mathrm{y}-l+q}{\sigma}\right) \\
&\leqslant \sum_{l=1}^{q/2-1}(-l)\phi\left(\frac{\mathrm{y}-l}{\sigma}\right) \qquad (24) \\
&\quad + \sum_{l=q/2+1}^{q-1}(q-l)\phi\left(\frac{\mathrm{y}-l+q}{\sigma}\right).
\end{aligned}
$$

Where (23) is obtained by simple rearrangement, and (24) follows from discounting non-positive terms in the sum. Next, manipulating the indices of the second sum of (24), and rewriting as a single sum we get

$$
\begin{aligned}
&= \sum_{l=1}^{q/2-1}\left[(-l)\phi\left(\frac{\mathrm{y}-l}{\sigma}\right)+l\phi\left(\frac{\mathrm{y}+l}{\sigma}\right)\right] \\
&= \sum_{l=1}^{q/2-1} l\left[\phi\left(\frac{\mathrm{y}+l}{\sigma}\right)-\phi\left(\frac{\mathrm{y}-l}{\sigma}\right)\right].
\end{aligned}
$$

Finally, $\mathrm{y}>0$ implies that the argument $(\mathrm{y}+l)/\sigma$ is larger in absolute value than $(\mathrm{y}-l)/\sigma$. Consequently, all the summands in the last sum are negative, and the posterior probability is monotone decreasing. □

REFERENCES

[1] A. Ahmed, R. Kötter, and N. Shanbhag, "VLSI architectures for soft-decision decoding of Reed-Solomon codes," in *International Conference on Communications*. IEEE, Jun. 2004, pp. 2584–2590.

[2] E. Berlekamp and R. McEliece, "Average-case optimized buffered decoders," in *The Impact of Processing Techniques on Communications*. Martinus Nijhoff publishers, The Netherlands: NATO Advanced science institutes, 1985, pp. 145–158.

[3] R. Blahut, *Theory and Practice of Error Control Codes*. Reading, Massachusetts: Addison-Wesley, 1983.

[4] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*. New York NY: Springer, 1997.

[5] P. Elias, "List decoding for noisy channels," Technical report, Research Laboratory of Electronics, MIT, Tech. Rep., 1957.

[6] G. Feng and K. Tzeng, "A generalization of the Berlekamp Massey algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes," *IEEE Transactions on Information Theory*, vol. 37, no. 5, pp. 1274–1287, Sep. 1991.

[7] W. Gross, F. Kschischang, R. Kötter, and P. Gulak, "Applications of algebraic soft-decision decoding of Reed-Solomon codes," *IEEE-Trans-Comm*, vol. 54, no. 7, pp. 1224–1234, 2006.

[8] V. Guruswami, "List decoding in average-case complexity and pseudo-randomness," in *Proc. IEEE Information Theory Workshop*, Punta del Este, Uruguay, 2006.

[9] V. Guruswami and M. Sudan, "Improved decoding of Reed-Solomon and algebraic-geometric codes," *IEEE Transactions on Information Theory*, vol. 45, pp. 1755–1764, 1999.

[10] R. Kötter, "Fast generalized minimum-distance decoding of algebraic-geometry and Reed-Solomon codes," *IEEE Transactions on Information Theory*, vol. 42, no. 3, pp. 721–736, 1996.

[11] ——, "Efficient bivariate interpolation," *Personal communication*, 2004.

[12] R. Kötter and A. Vardy, "Algbraic soft-decision decoding of Reed-Solomon codes," *IEEE Transactions on Information Theory*, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.

[13] ——, "A complexity reducing transformation in algebraic list decoding of Reed-Solomon codes," in *Proc. IEEE Information Theory Workshop*, Paris, France, 2003.

[14] R. McEliece, "The Guruswami-Sudan algorithm for decoding Reed-Solomon codes," JPL, http://www.ipnpr.jpl.nasa.gov/progress_report/42-153/, Tech. Rep. IPN progress report 42-153, 2003.

[15] R. Nielsen and T. Høholdt, "Decoding Reed-Solomon codes beyond half the minimum distance," *Cryptography and Related Areas, J. Buchmann et al. eds. Springer-Verlag 2000*, pp. 221–236, 2000.

[16] R. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance," *IEEE Transactions on Information Theory*, vol. 46, no. 1, pp. 246–257, 2000.

[17] M. Sudan, "Decoding of Reed-Solomon codes beyond the error correction bound," *J. Complexity*, vol. 12, pp. 180–193, 1997.

[18] ——, "List decoding: Algorithms and applications," *SIGACT News*, vol. 31, 2000.

[19] S. Tang, L. Chen, and X. Ma, "Progressive list-enlarged algebraic soft decoding of Reed-Solomon codes," *IEEE Communications Letters*, vol. 16, no. 6, pp. 901–904, 2012.

[20] L. Trefethen and D. Bau, *Numerical Linear Algebra*. Philadelphia PA: Society for Industrial and Applied Mathematics, 1997.

[21] J. Wozencraft, "List decoding," Quarterly Progress Report, Research Laboratory of Electronics, MIT, Tech. Rep. 48, 1958.

Yuval Cassuto (S'02-M'08) is a faculty member at the Department of Electrical Engineering, Technion, Israel Institute of Technology. His research interests lie at the intersection of the theoretical infomration sciences and the engineering of practical computing and storage systems.

During 2010-2011 he has been a Scientist at EPFL, the Swiss Federal Institute of Technology in Lausanne.

From 2008 to 2010 he was a Research Staff Member at Hitachi Global Storage Technologies, San Jose Research Center.

He received the B.Sc degree in Electrical Engineering, summa cum laude, from the Technion, Israel Institute of Technology, in 2001, and the MS and Ph.D degrees in Electrical Engineering from the California Institute of Technology, in 2004 and 2008, respectively.

From 2000 to 2002, he was with Qualcomm, Israel R&D Center, where he worked on modeling, design and analysis in wireless communications.

Dr. Cassuto has won the 2010 Best Student Paper Award in data storage from the IEEE Communications Society, as well as the 2001 Texas Instruments DSP and Analog Challenge $100,000 prize.

Jehoshua Bruck (S'86-M'89-SM'93-F'01) received the B.Sc. and M.Sc. degrees in electrical engineering from the Technion-Israel Institute of Technology, Haifa, Israel, in 1982 and 1985, respectively, and the Ph.D. degree in electrical engineering from Stanford University, Stanford, CA, in 1989.

He is the Gordon and Betty Moore Professor of computation and neural systems and electrical engineering at the California Institute of Technology, Pasadena, CA. His extensive industrial experience includes working with IBM Almaden Research Center, as well as co-founding and serving as Chairman of Rainfinity, acquired by EMC in 2005; and XtremIO, acquired by EMC in 2012. His current research interests include information theory and systems and the theory of computation in biological networks.

Dr. Bruck is a recipient of the Feynman Prize for Excellence in Teaching, the Sloan Research Fellowship, the National Science Foundation Young Investigator Award, the IBM Outstanding Innovation Award, and the IBM Outstanding Technical Achievement Award. His papers were recognized in journals and conferences, including winning the 2010 IEEE Communications Society Best Student Paper Award in Signal Processing and Coding for Data Storage for his paper on codes for limited-magnitude errors in flash memories, the 2009 IEEE Communications Society Best Paper Award in Signal Processing and Coding for Data Storage for his paper on rank modulation for flash memories, the 2005 A. Schelkunoff Transactions Prize Paper Award from the IEEE Antennas and Propagation Society for his paper on signal propagation in wireless networks, and the 2003 Best Paper Award in the Design Automation Conference for his paper on cyclic combinational circuits.

Robert J. McEliece (M'70-SM'81-F'84-LF'08) Robert J. McEliece was born in Washington DC in 1942. He graduated from the Baltimore Polytechnic Institute in June, 1960 and enrolled as a Caltech freshman in September, 1960.

In June 1963, he was hired by Dr. Solomon Golomb as a part-time math assistant at the Communications Research Division, at Jet Propulsion Laboratory. He completed his graduate studies in mathematics under the supervision of Prof. Marshall Hall, Jr. in 1967.

In the same year he converted to full time at JPL where he worked on coding theory until 1978. From 1978 to 1982 he was a Professor of mathematics at the University of Illinois, Urbana-Champaign. From 1982 to 2007, he was a Professor of Electrical Engineering at Caltech and was Executive Officer for EE from 1990 to 1999. From 1997 2007, he was also the Allen E. Puckett Professor at Caltech.

He retired in January, 2008 which is 47 years and 3 months since he first set foot on California soil.