After this paper was published, its algorithms were implemented by David Kohl, who found several misprints in the original published version of the paper. The following is the slightly revised version in which those misprints have been corrected.

Bounded Distance + 1 Soft-Decision Reed–Solomon Decoding

Elwyn Berlekamp, *Fellow, IEEE*

***Abstract*— We present a new Reed–Solomon decoding algorithm, which embodies several refinements of an earlier algorithm. Some portions of this new decoding algorithm operate on symbols of length $lg q$ bits; other portions operate on somewhat longer symbols. In the worst case, the total number of calculations required by the new decoding algorithm is proportional to nr, where n is the code's block length and r is its redundancy. This worst case workload is very similar to prior algorithms. But in many applications, average-case workload and error-correcting performance are both much better. The input to the new algorithm consists of n received symbols from GF(q), and n nonnegative real numbers, each of which is the reliability of the corresponding received symbol. Any conceivable errata pattern has a "score" equal to the sum of the reliabilities of its locations with nonzero errata values. A max-likelihood decoder would find the minimum score over all possible errata patterns. Our new decoding algorithm finds the minimum score only over a subset of these possible errata patterns. The errata within any candidate errata pattern may be partitioned into "errors" and "erasures," depending on whether the corresponding reliabilities are above or below an "erasure threshold." Different candidate errata patterns may have different thresholds, each chosen to minimize its corresponding ERRATA COUNT, which is defined as**

$$2 \cdot (\text{number of errors}) + (\text{number of erasures}).$$

The new algorithm finds an errata pattern with minimum score among all errata patterns for which

$$\text{ERRATA COUNT} \le r + 1$$

where r is the redundancy of the RS code. This is one check symbol better than conventional RS decoding algorithms. Conventional algorithms also require that the erasure threshold be set *a priori*; the new algorithm obtains the best answer over all possible settings of the erasure threshold.

Conventional cyclic RS codes have length $n = q - 1$, and their locations correspond to the nonzero elements of GF(q). The new algorithm also applies very naturally to RS codes which have been doubly extended by the inclusion of 0 and ∞ as additional locations. If the actual code length is less than $q + 1$, then a very minor simplification can be obtained by selecting ∞ as one of the omitted symbols. However, even in this case, the decoder can improve performance by transforming the received word to include the location ∞ before decoding, and then reversing the transformation after the errata pattern has been found. The new algorithm also provides a foundation from which a talented designer can tailor the algorithm to better fit any of a wide variety of particular system requirements. We discuss some of these opportunities to obtain further significant improvements by appropriate custom engineering.

Manuscript received August 31, 1994; revised August 14, 1995. The material in this paper was presented in part at the IEEE International Symposium on Information Theory, San Antonio, TX, January 1993 (Shannon Lecture).

The author is with Hazelnut, Berkeley, CA 94704 USA.

Publisher Item Identifier S 0018-9448(96)01688-4.

***Index Terms*—Decoding algorithms, Reed–Solomon codes, soft decisions, doubly extended codes, algebraic decoding, decoding beyond half minimum distance, probabilistic decoding, best case decoding, queued decoding, Generalized Minimum Distance decoding, bounded distance decoding.**

I. Historical Background

A. Intended Range of Code Parameters

SEVERAL parameters are required to specify a particular Reed–Solomon code. The algorithms we discuss in this paper are intended for use in cases such as the following:

System	q	n	r	Channel Speed	Date
Cinema Digital Sound	64	63	21	4 Mb/s	1988
"Hypersystolic" [9]	64	63	12	820 Mb/s	1987
NASA Space Telescope	256	240	16	1 Mb/s	1980
NASA Deep Space	256	255	32	varied	varied
Longitudinal Mag Tape	256	34	6	68 Mb/s	1984

Notice that this table does not include the RS codes most commonly used on compact-disk music players, nor on magnetic disk systems. Despite their extremely wide usage, most current implementations of most such codes correct at most one or two errors per block, and in some cases only two erasures per block. In such degenerate cases, the relative advantages of the algorithms presented in this paper can disappear. Nor do we recommend the new algorithms for extremely long RS codes, with alphabet sizes much larger than 2^8. As the symbol size approaches infinity, better asymptotic algorithms are known [15].

Furthermore, as indicated by the table, our primary interest is in Reed–Solomon codes over fields of characteristic two. The generalization to fields of odd characteristic involves little more than taking care to distinguish between plus and minus signs.

B. Forney's Early Work

In 1965, Forney [12] introduced a scheme for decoding Reed–Solomon codes with soft-decision information. He called this scheme "Generalized Minimum Distance Decoding." In Forney's model, each received symbol was accompanied by a nonnegative real number, called "*score*[y_j]," which measures its reliability. A symbol with higher score is more likely to be correct than another symbol with lower score.

We suppose the code has length n, redundancy r, and distance $d = r + 1$. To decode, Forney began by sorting to find a list of the r least reliable received symbols. Rather than initially erasing all received symbols whose scores were below any fixed, preset erasure threshold, Forney iterated over all possible values of the erasure threshold. This entailed $(d+1)/2$ separate passes through the full decoding program. For $t = 0, 1, \cdots, (d-1)/2$, on the tth such pass, Forney began by erasing the $(r - 2t)$ least reliable characters. A conventional RS decoding program then decoded to find the unique candidate errata pattern with no more than t errors, if any such pattern existed. Each candidate error pattern was then assigned a *"SCORE"* equal to the sum of the *scores* of its errors and its unreadable erasures. The best candidate was chosen by comparing these SCORE's. Finally, the candidate errata pattern with the lowest SCORE was selected as the winner.

Forney's performance analysis showed that this scheme often yielded better results than could be achieved by using any fixed preset value of the erasure threshold. But Forney's analysis also showed that the true, max-likelihood error pattern often corresponds to t errors and s erasures, where $2t+s \geq d$. Hence, "bounded distance" decoding still falls substantially short of optimal, because the traditional RS decoding methodologies work only when $2t + s < d$. And the workload of the original "bounded distance" algorithm was roughly $d/2$ times as large as the workload of a conventional RS decoder with a fixed erasure threshold. To see how we can now eliminate this extra factor of $d/2$, and simultaneously achieve a significantly lower probability of block decoding error, it is useful to review the historical development of traditional algebraic decoding algorithms using fixed erasure thresholds. The simplest and most popular special case is the degenerate one in which the erasure threshold is set so high that every received symbol's score is quantized to zero. This is the case of no erasures, or "errors-only" decoding.

C. Traditional RS Decoding Algorithms

The block length and the distance of an RS code are design parameters which can vary over a wide range, and a similarly wide variety of decoding algorithms has been proposed. When the code rate is sufficiently low, or the block length is sufficiently long, then transform decoding [10] or special asymptotic techniques [15] merit serious consideration. But, in one range of considerable interest, including $n = 255$, $r = 32$; or $n = 255$, $r = 16$; or $n = 32$, $r = 6$, the "traditional" algebraic RS decoders are still the most popular. They typically include the following subprograms:

1) Re-encode to find the syndromes.
2) Transform the equations to account for the erasures.
3) Find the error-locator polynomial.
4) Find the roots of the error-locator polynomial.
5) Find the error values and the erasure values.

The workload required by steps 2), 3), and 5) is easily overbounded by a term proportional to r^2, but steps 1) and 4) entail a workload proportional to rn. Hence, in the common case in which the code rate is significantly greater than $\frac{1}{2}$, subprograms 1) and 4) entail more computation than any of the others.

During the last three decades, there has been a series of continual improvements in several of the subprograms for RS decoding [4], [9], [20], [22], [23], [25], [28], [29]. Many of the subprograms' benefits are conspicuous in some system environments and negligible in others. For example, in many memory systems (including magnetic tapes for instrumentation recording, as well as optical and magnetic disks for computer memory systems), the very high data reliability requirements typically drive decoders to operate in a region of relatively "safe" raw error rates. In this region, the actual number of errors per block, e, is usually much less than $r/2$. Even though system requirements demand that the decoder be capable of correcting the maximum number of errors, throughput may be limited instead by "average" rather than "worst case" error statistics. Under such circumstances, the relative workload of subprogram 4) diminishes significantly in comparison to subprogram 1). Partly, this is due to the obvious fact that it is proportional to en rather than to rn. Further big improvements are possible by using special techniques to find roots of polynomials of low degree [5]. For these reasons, in the late 1970's, the costs of RS decoders for some memory systems was dominated by subprogram 1). But then the introduction of "Bit-Serial RS Decoders" [8] yielded such a large reduction in the coefficient of the workload associated with this subprogram that subprogram 3) again became the dominant term in this system environment. Welch and Berlekamp then broke that bottleneck by introducing a new algorithm [22], [29] for subprogram 3). This algorithm over finite fields has some similarities with classical "Padé approximations" [2], [3] over real and complex fields. The details of this algorithm for the "errors-only" case are presented in Section II-A and Appendix I.

D. Other Recent Work

Several years after the appearance of Welch–Berlekamp [29], several other authors have independently derived other decoding schemes, each of which achieves some nontrivial proper subset of the performance advantages presented in this paper. Kotter [18] finds a subspace containing all the possible solutions and gives an algorithm to find the vector with zeros in the erasure locations. It appears to require more than one run to achieve GMD. Sorger [27] gets a new RS decoding algorithm with a one-run GMD decoder. This decoding algorithm still uses power-sum symmetric functions, on which it does Newton interpolation. Morii and Kasahara [22] derive a different form of key equation, which, like Welch–Berlekamp [29] avoids the computation of the power-sum symmetric functions. Araki, Takada, and Morii [1] give a one-run GMD errors and erasures procedure. They define a more generalized syndrome polynomial which reduces back to

Welch–Berlekamp [29] in the cases of interest in the present paper.

II. Welch–Berlekamp Decoding

A. Origin of the Equations for Errors-Only Decoding

Re-encoding reveals one possible error pattern directly. This is the error pattern which assumes that all message symbols are correct and that all errors occurred in the check symbols. The error values of this "syndrome" error pattern are denoted by $s[y_1], s[y_2], \cdots, s[y_r]$. Here y_i runs only over the code's r check locations. Long ago, Slepian [26] observed that the difference between any two error patterns in the same coset must be a codeword. All variations of Welch–Berlekamp decoding concentrate on finding a codeword which will translate some known possible error pattern into another more likely error pattern in the same coset. In the original, simplest case, we have no reliability information at all. The one known error pattern is the syndrome, and the most likely error pattern is Slepian's coset leader, which has minimum weight. The Welch–Berlekamp algorithm will find the codeword which translates the coset into an error pattern of Hamming weight $\leq r/2$, if any such exists.

The following structural theorem provides the foundation for the equations that the Welch–Berlekamp algorithm solves. If $f(z)$ is a polynomial, we denote its derivative by f' and its degree by $|f|$.

In each application of Theorem 1, the roots of f are the locations of the nonzero components NOT of the error pattern, but of a codeword. This codeword is NOT the transmitted codeword, but rather any of several other codewords (or the sum of such codewords) by which the actual error pattern differs from other more easily recognizable members of its coset.

Theorem 1 (Characterization of Codewords in Shortened RS Codes): Let $\mathcal{F}$ be a field, and let $\mathcal{S}$ be any subset of distinct elements of $\mathcal{F}$.

Let

$$f(z) = f_{|f|} \prod_{y \in \mathcal{S}} (z - y)$$

a polynomial whose leading coefficient $f_{|f|}$ is nonzero.

Let $N(z)$ be a polynomial whose degree is $|N|$, where

$$|N| \leq |f| - r.$$

Let

$$C[\infty] = \frac{-N_{|f|-r}}{f_{|f|}}$$

(If $|N| = |f| - r$, then $N_{|f|-r}$ is the leading coefficient of N; otherwise, $C[\infty] = 0$).

Let

$$C[y] = \frac{N(y)}{f'(y)}, \quad \text{for all } y \in \mathcal{S}.$$

Then

$$C[\infty] + \sum_{y \in \mathcal{S}} y^{r-1} C[y] = 0$$

and

$$\sum_{y \in \mathcal{S}} y^j C[y] = 0, \quad \text{for } j = 0, 1, 2, \cdots, r-2.$$

Comments: Our primary interest is the RS code over the field $\mathcal{F} = \mathrm{GF}(2^m)$, with generator polynomial

$$g(z) = \prod_{i=0}^{r-1} (z - \alpha^i).$$

The distance of this code is $d = r + 1$. Given any set $\mathcal{S}$ of $|f|$ locations, and any polynomial N of degree $< |f| - r$, the theorem asserts that there is a codeword whose coefficients are given by

$$C[y] = \begin{cases} 0, & \text{if } f(y) \neq 0 \\ \dfrac{N(y)}{f'(y)}, & \text{if } f(y) = 0. \end{cases}$$

Since the RS code is well known to be maximal-distance-separable, the dimension of the subcode whose nonzero coefficients all lie in $\mathcal{S}$ is $|f| - r$. Every nonzero RS codeword must be of the specified form, for an appropriate choice of $f(z)$ and $N(z)$.

Proof: Consider the rational function $\frac{z^j N(z)}{f(z)}$. A partial fraction decomposition gives

$$\frac{z^j N(z)}{f(z)} = \sum_{y_i \in \mathcal{S}} \frac{a_i}{z - y_i}. \tag{1}$$

To obtain the value of the scalar a_i, we multiply (1) by $f(z)$, and then set $z = y_m$ to obtain

$$\begin{aligned} y_m^j N(y_m) &= \sum_{y_i \in \mathcal{S}} a_i \prod_{k \neq i} (y_m - y_k) \\ &= a_m \prod_{k \neq m} (y_m - y_k) \\ &= a_m f'(y_m) \end{aligned}$$

whence

$$a_m = \frac{y_m^j N(y_m)}{f'(y_m)}. \tag{2}$$

Substituting (2) into (1) and multiplying by z gives

$$\frac{z^{j+1} N(z)}{f(z)} = \sum_{y_i \in \mathcal{S}} \frac{y_i{}^j N(y_i)}{f'(y_i)} \frac{z}{(z - y_i)}. \tag{3}$$

Define the reciprocal polynomials

$$\tilde{N}(x) = x^{|N|} N(x^{-1})$$
$$\tilde{f}(x) = x^{|f|} f(x^{-1})$$

so that, with $z = x^{-1}$, (3) becomes

$$\frac{x^{|f|-|N|-j-1} \tilde{N}(x)}{\tilde{f}(x)} = \sum_{y_i \in \mathcal{S}} \frac{y_i{}^j N(y_i)}{f'(y_i)(1 - y_i x)}. \tag{4}$$

Setting $x = 0$ gives $\tilde{f}(0) = f_{|f|}$, $\tilde{N}(0) = N_{|N|}$, and

$$\sum_{y_i \in \mathcal{S}} \frac{y_i{}^j N(y_i)}{f'(y_i)} = \begin{cases} 0, & \text{if } j + 1 < |f| - |N| \\ \frac{N_{|N|}}{f_{|f|}}, & \text{if } j + 1 = |f| - |N|. \end{cases}$$

If $r = |f| - |N|$, the theorem follows directly.

If $r < |f| - |N|$, then $N_{|f|-r} = 0$, $C[\infty] = 0$, and the theorem again follows. □

Corollary 1: If $\vec{C}$ is a minimum-weight Reed–Solomon codeword, whose nonzero coordinates lie in the set $\mathcal{S}$, where $\mathcal{S} \subset \mathcal{F}$, $|\mathcal{S}| = r+1$, then, to within a scalar multiple N_0, $\vec{C}$ may be expressed as follows:

$$C[y] = \begin{cases} 0, & \text{if } y \notin \mathcal{S} \\ \prod_{\xi \in \mathcal{S}-\{y\}} (y-\xi)^{-1}, & \text{if } y \in \mathcal{S}. \end{cases}$$

The original case of interest is errors-only decoding, with no erasures. In this case, it is convenient to let $\vec{C}$ be the codeword which is the difference between the error pattern and the syndrome of the re-encoded received word, whose nonzero coordinates all lie in the code's check locations. In this case, $f(z)$ has two factors:

$$f(z) = F(z)W(z)$$

where $W(z)$ is the (unknown) error-locator polynomial, whose roots will be the error locations, and $F(z)$ is the known check-locator polynomial,

$$F(z) = \prod_{i=1}^{r}(z - y_i).$$

The translating codeword is the difference of the syndromes and the error pattern. If there are erroneous checks, $F(z)$ and $W(z)$ will have roots in common, but we anticipate that such roots will also appear in the numerator polynomial $N(z)$.

We require the value of the translating codeword to agree with the syndrome at each check location which is not also a root of $W(z)$ and $N(z)$. This gives these equations, for $i = 1, 2, \cdots, r$

$$N(y_i) = (FW)'(y_i)s[y_i].$$

But $(FW)' = FW' + WF'$, and since y_i is a check location, $F(y_i) = 0$ and we have

$$N(y_i) = W(y_i)F'(y_i)s[y_i] \quad (5)$$

As i ranges from 1 to r, this yields r equations (5) in the unknown coefficients of N and W. These equations (5) are called the Welch–Berlekamp equations.

B. Lemmas Leading to the WB Algorithm

Impatient readers may prefer to skip this section and jump ahead to the algorithm in Appendix I. However, there are a few lemmas that we will need later that do not depend on the algorithm. Since they provide some motivation for parts of the algorithm, we present them now.

The *problem* is now specified by a set of j pairs of input data points: $x_1, x_2, \cdots, x_j$ and $y_1, y_2, \cdots, y_j$, all of which are elements of a field $\mathcal{F}$. A *solution* is a pair of polynomials, N and W, such that for $i = 1, 2, \cdots, j$

$$N(y_i) = W(y_i)x_i.$$

We let z denote the indeterminate, and write N for $N(z)$, W for $W(z)$. We refer to a solution as "N/W" even though this refers to the pair of polynomials, not to their quotient, which in this paper we denote by the horizontal bar, $\frac{N}{W}$. Suppose that N and W have a common factor, f, so that

$$\begin{aligned} N &= nf \\ W &= wf \end{aligned}$$

We call such a common factor *removable* if n/w is also a solution. The reader should notice that, for example, a common factor of the form $(z - y_i)$ might or might not be removable, and the only common factors which might not be removable are of that form. We call a solution which has no removable common factors *irreducible*. An irreducible solution *may* have one or more common factors of the form $(z - y_i)$.

We define the *rank* of a solution N/W as

$$\text{Rank}(N/W) = \max\{2|W|, 1 + 2|N|\}.$$

We call a solution *monic* if W is monic and $|W| > |N|$, or if N is monic and $|N| \geq |W|$.

Lemma 1: If N/W is an irreducible solution and M/V is another solution, such that

$$\text{Rank}(N/W) + \text{Rank}(M/V) \leq 2j$$

then M/V can be reduced to N/W by canceling removable factors.

Lemma 2: If N/W is an irreducible monic solution of rank $\leq j$, and if M/V and L/U are two monic irreducible solutions such that

$$\text{Rank}(L/U) = \text{Rank}(M/V) = 2j + 1 - \text{Rank}(N/W)$$

then there exists a polynomial g such that

$$\begin{aligned} L &= M - gN \\ U &= V - gW. \end{aligned}$$

Proofs of Lemmas 1 and 2: Let N/W and M/V be two solutions, and consider the polynomial $MW - NV$.

Its roots include $y_1, y_2, \cdots, y_j$, so it must be a multiple of

$$\prod_{i=1}^{j}(z - y_i).$$

Let

$$\begin{aligned} \text{Rank}(N/W) &= R_1 \\ \text{Rank}(M/V) &= R_2 \end{aligned}$$

then

$$\begin{aligned} 1 + 2|N| &\leq R_1 \\ 2|W| &\leq R_1 \\ 1 + 2|M| &\leq R_2 \\ 2|V| &\leq R_2 \end{aligned}$$

$$|MW - NV| \leq \max\{|MW|, |NV|\} = \frac{R_1 + R_2 - 1}{2}.$$

Furthermore, if $R_1 + R_2$ is odd, and if N/W and M/V are both monic, then either $MW - NV$ or $NV - MW$ must also be monic.

We have seen that $|MW-NV| < \frac{R_1+R_2}{2}$. In the hypothesis of Lemma 1, we also have $\frac{R_1+R_2}{2} \le j$, whence $|MW - NV| < j$. Since $MW - NV$ is a multiple of a polynomial of degree j, it must be identically zero, and we have

$$MW = NV$$

Let $d(z)$ be the greatest common divisor of $W(z)$ and $V(z)$. Then $\frac{W}{d}$ divides N, $\frac{V}{d}$ divides M, and the two quotients are the same polynomial which we call h

$$\frac{W}{d} = \frac{N}{h}; \frac{V}{d} = \frac{M}{h}.$$

We now claim that h/d is also a solution. If not, then there must be some y_i for which $d(y_i) \neq 0$ and

$$h(y_i) \neq d(y_i)x_i.$$

Yet, since $N(y_i) = W(y_i)x_i$, it follows that $(z - y_i)$ must divide $\frac{W}{d}$. Similarly, since $M(y_i) = V(y_i)x_i$, it also follows that $(z-y_i)$ must divide $\frac{V}{d}$. But this contradicts the definition of d as the greatest common divisor of W and V.

So h/d is also a solution, and both N/W and M/V can be reduced to it by canceling out removable factors, if any exist. This completes the proof of Lemma 1. □

To prove Lemma 2, we observe that the difference of two monic solutions of the same rank is a solution of lower rank. So Lemma 1 applies to $(L - M)/(U - V)$ which must be gN/gW, for some removable factor g. □

Definition: We call a pair of monic solutions N/W and M/V *complementary* if

$$\text{Rank}\,(N/W) + \text{Rank}\,(M/V) = 2j + 1$$

and if

$$MW - NV = \pm \prod_{i=1}^{j}(z - y_i).$$

Lemma 3: If N/W and M/V are complementary, then W and V are relatively prime.

Proof: Any common factor would necessarily divide

$$MW - NV = \prod(z - y_i).$$

But if some $(z - y_i)$ divides W then $W(y_i) = 0$, whence $N(y_i) = 0$. Similarly, the fact that $(z - y_i)$ divides V implies $V(y_i) = 0$, whence $M(y_i) = 0$. So if $(z - y_i)$ is a common factor of W and V, then $(z - y_i)^2$ divides $MW - NV$, a contradiction. □

A consequence of the lemmas is that there can never be more than one irreducible solution of rank $\le j$. Our algorithm will always find such a solution, called N/W, thereby proving that the solution is not only unique, but that it exists. Our algorithm also calculates another pair of polynomials which is closely related to a complementary solution. When our algorithm was first used in the early 1980's, the polynomials M and V were seen only as stepping stones to obtain the error-locator polynomial,[1] $W = W^{(r)}$ and the polynomial $N = N^{(r)}$. After a subsequent subprogram finds the roots of W, another subprogram uses N to find the values of the errors.

[1] The reader may notice many analogies between the present W, N, V, M, and the respective polynomials $\sigma, \omega, \tau, \gamma$, which appeared in [4, ch. 7].

The property of this algorithm which we first found most attractive was the fact that if there were only e errors, and $e << \frac{r}{2}$, then the workload would be proportional to ed rather than to d^2.

It was not easy to formulate an algorithm which achieves this advantage. To do so, some computations which are necessary when there are many errors must be evaded when there are few errors. Unfortunately, at the logical time for doing these computations, the algorithm's knowledge is insufficient to ensure a correct decision about whether or not the computations will need to be done. If these computations are done, they may later prove unnecessary, and such unnecessary computations can exceed the workload budget. If instead the algorithm decides not to do the additional computation, it may later prove impossible to get the correct answers. This dilemma is resolved by introducing a queue, which holds the troublesome cases. If subsequent calculations prove the existence of more errors, then the calculations relating to the next element in the queue are performed and the queue is shortened. However, if subsequent data confirm that there are few errors, then the algorithm terminates with a large queue of potential work that never needed to be done.

Attempts to provide any more detailed explanation of the contents of the queue have proved unproductive. The queue is simply a convenient gimmick to store elements representing work which can be deferred, possibly forever.

The algorithm is stated in Appendix I.

C. Detailed Assertions and Sketched Proofs

This section proves properties of the algorithm stated in Appendix I.

Unless otherwise stated, each assertion is applicable both at line 2 and at line 30. Some assertions are also applicable between lines 14 and 15.

A1: If $i \le j$
and $y = rcheck[i]$
then
$N^{(j)}(y) = W^{(j)}(y)F'(y)s[y]$
and $M^{(j)}(y) = V^{(j)}(y)F'(y)s[y]$

A2: If $qin \ge i > qout$
and $y = queue[i]$
then $N^{(j)}(y) = W^{(j)}(y)F'(y)s[y]$

Proof of A1 and A2: Also include A1 at line 15 and the assertion on line 20, and then prove all claims by a straightforward induction, noticing that the hypothesis of A2 forces success of the conditional on line 15, so that if lines 10-13 were executed since the prior verification of this assertion, then so were lines 23–26.

A3: $|W^{(j)}| \le \lfloor\frac{j}{2}\rfloor$
$|V^{(j)}| \le \lceil\frac{j}{2}\rceil$

Proof of A3: Use induction at line 10, or

$$|W^{(j+1)}| \le \max\left\{\left\lceil \frac{j}{2}\right\rceil, \left\lfloor \frac{j}{2}\right\rfloor\right\} = \left\lceil \frac{j}{2}\right\rceil = \left\lfloor \frac{j+1}{2}\right\rfloor.$$

Similarly, at line 12

$$|V^{(j+1)}| \le 1 + \left\lfloor \frac{j}{2}\right\rfloor = \left\lceil \frac{j+1}{2}\right\rceil.$$

A4: $|N^{(j)}| \le \left\lfloor \frac{j-1}{2}\right\rfloor$

$|M^{(j)}| \le \left\lceil \frac{j-1}{2}\right\rceil$

Proof of A4: This is analogous to proof of A3.

A5: If j is even, then $|W^{(j)}| = \left\lfloor \frac{j}{2}\right\rfloor$

and $|M^{(j)}| = \left\lceil \frac{j-1}{2}\right\rceil$

If j is odd, then $|V^{(j)}| = \left\lceil \frac{j}{2}\right\rceil$

and $|N^{(j)}| = \left\lfloor \frac{j-1}{2}\right\rfloor$

A6: If j is even, $W^{(j)}$ and $M^{(j)}$ are monic

If j is odd, $N^{(j)}$ and $V^{(j)}$ are monic

Proofs of A5 and A6: These are done directly by a straightforward induction.

Theorem 2 (N/W Solves and M/J Complements): Let $y_i = rcheck[i]$. Then $N^{(j)}/W^{(j)}$ is the unique irreducible solution of rank $\le j$ of the equation

$$N^{(j)}(y_i) = W^{(j)}(y_i)F'(y_i)s[y_i], \quad \text{for } i \le j$$

and $M^{(j)}/V^{(j)}$ is a complementary solution.

Theorem 3 (Solution Survives the Reordering): Let $y_i = ocheck[i]$, for $i = 1, 2, \cdots, r$, and let j be whatever value is determined by the algorithm at its completion. Then $N^{(j)}/W^{(j)}$ is the unique irreducible solution of rank $\le \frac{r}{2}$ of the equation

$$N^{(j)}(y_i) = W^{(j)}(y_i)F'(y_i)s[y_i], \quad \text{for } i \le r.$$

Proof: The algorithm partitions

$$\{ochecks\} = \{rchecks\} + \{yqueue\}$$

with $j = |\{rchecks\}| \le r$. So if $y \in \{ochecks\}$, then either $y \in \{rchecks\}$ or $y \in \{yqueue\}$, and in either case, we have

$$N^{(j)}(y) = W^{(j)}(y)F'(y)s[y]. \qquad \square$$

In the case in which there are no erasures and t errors, one can choose $\{ochecks\}$ to be the r check symbols of the RS code, and then apply the WB algorithm. Theorems 1 and 3 show that if $t \le r/2$, then the polynomial $W^{(}j)(z)$ with which this algorithm terminates is the error locator polynomial, whose roots are the locations of the errors.

III. Application to Forney's Channel

A. The Grand Plan

One can also apply the original WB algorithm to Forney's channel, which provides a real-number reliability score with each received symbol. The appropriate grand plan to decode remains remarkably similar to the traditional plan mentioned in Section I-C. The new plan is as follows:

0) Sort the reliability scores.
1) Re-encode via bit-serial RSE.
2) Erase the least reliable r symbols, and decode using a traditional "erasures-only" algorithm.
3) Treat the least reliable r symbols as check symbols. Use the WB algorithm to process them in order (most reliable first, and least reliable last) to find the candidate solutions, $N^{(1)}/W^{(1)}, N^{(2)}/W^{(2)}, \cdots, N^{(r)}/W^{(r)}$.
4) For each candidate solution, find the errata pattern corresponding to $N^{(i)}/W^{(i)}$, if any, and compute the SCORE of that errata pattern.
5) Select the candidate errata pattern with the lowest score, and complete the decoding.

Subprogram 0 can be accomplished by any of many known sorting algorithms [17] which have workloads proportional to $n \log n$. Unless the code rate is extremely high, this will be dominated by the workloads of subsequent subprograms.

Subprogram 1 has a workload proportional to nr, but the constant of proportionality is extremely small [8], [24]. Indeed, contrary to popular folklore, RS encoders are significantly *simpler* than encoders for most binary codes having the same amount of redundancy.

Subprogram 2 has a workload proportional to r^2, using any of several traditional decoding methods [4], [28], [29], or a refined variation we shall present in Section VI-F.

Subprogram 3 now has a workload proportional to re, where e is the actual number of errata. This is a big improvement over prior algorithms in two respects: First, it finds *all* candidate solutions in a single pass through the data, with an amount of computation that is comparable to what prior algorithms required for only the single (worst) case in which there were no erasures. Second, when e is significantly less than $\frac{r}{2}$, the algorithm terminates with many symbols in the queue and with $j << r$. The workload is seen to be proportional to re rather than to r^2.

Subprogram 4 apparently requires a workload proportional to ni for the ith candidate, yielding a total proportional to ne^2. In the worst case, when e is near $\frac{r}{2}$, this term dominates the workload. We will improve this substantially in Section IV, which also gives more details on Subprogram 5.

All of these subprograms were implemented in 1983-1984 [29] in a decoder that corrected almost all patterns of up to five concurrent dropouts on a 34-track longitudinal magnetic tape machine operating continuously at 68 Mb/s. Only six of the 34 tracks carried redundancy; the others recorded information. The code was a shortened RS code over GF (256), with $n = 34$ and $r = 6$. The soft-decision reliability information was derived from monitoring the recent history of errors, and this form of bounded distance decoding attained a higher level of performance than anyone had previously considered feasible. In that application, the reliability scores changed so slowly that their order was usually unchanged from one block to the next. This meant that whatever erasure threshold had been used on the prior block would very likely remain optimal for the present block, so that the computation of multiple scores by Subprogram 4 occurred so rarely that it had no noticeable effect on the average workload. All received data passed through a constant-length buffer which was sufficiently long that performance was more nearly related to average-case workloads rather than worst case workloads. Buffer overflows were prevented by a "punting" strategy which allowed undecoded data to pass through the system when the decoder ran too far behind the channel. The total bit error rate in the data delivered to the user by the decoder was better than 10^{-9}, and most of it was due to undecodable error patterns rather than to punts.

B. Scoring for Bounded-Distance Decoding

For each value of t, $N^{(2t)}/W^{(2t)}$ represents a potential candidate solution to the known equations. This solution may correspond to an errata pattern with t errors, located at the t roots of $W^{(2t)}$, and $s = r - 2t$ erasures, located in the code's s least reliable locations.

To compute the SCORE of this candidate errata pattern, the algorithm in Appendix II begins with a crude initial approximation that ignores N and W and uses only the values of s and t. This initial approximation is conservative in two respects. First, it assumes that all of the s erased symbols are incorrect, and it accounts for that by adding up their scores. Secondly, it assumes that all t errors occur in locations that all have the same high score, called "BIG." The algorithm then examines the candidate errata pattern at each of the code's n locations, and makes the appropriate adjustments in the score. These adjustments are of two types:

Readable Erasure:

This is an erased location where the value of the errata pattern is zero. Its score needs to be subtracted from the errata pattern's total SCORE.

Error:

When an error is located, its score replaces the default value of "BIG" in the total SCORE.

Recall that the N/W polynomials computed by the WB algorithm characterize the codeword which translates the syndrome into the actual errata pattern. In bounded distance decoding, the syndrome as seen by the WB algorithm is the errata pattern which is confined to the code's least reliable r locations. This errata pattern was computed by some "erasures-only" decoding algorithm. But the WB algorithm views this errata pattern as the "syndrome," and calls its locations the original checks, *ocheck*[ ]. While computing the N's and W's, the WB algorithm reorders these *ochecks* into the more convenient *rchecks*. Theorem 3 ensures that erasing the last s *rchecks* works just as well as erasing the last s *ochecks*.

The errata pattern found by the WB algorithm is the sum of the syndrome and the translating codeword. Since the syndrome may have nonzero values at check locations but not at message locations, the SCORING algorithm must split the errors into two cases to accommodate the difference between the formula for values of check errors and the formula for values of message errors.

This algorithm appears in Appendix II.

Theorem 4 (Scoring Algorithm Works; Errata Values): If SCORE $\geq$ BIG, then there is no errata pattern with the following properties:

1) The number of errata locations among *ocheck*[i] for $i > 2t$ is denoted by s.
2) It contains t errata among all other locations.
3) $2t + s \leq r = d - 1$.

If SCORE < BIG, then there is a unique errata pattern with these properties. Its SCORE is the computed value, and its values may be found as follows:

```
for $y \in \{yqueue\}$
    $errata[y] = 0$
for $y \in \{rcheck\}$
    if $(W^{(2t)} \neq 0)$ $errata[y] = 0$;
    else
        $errata[y] = \dfrac{-N^{(2t)}(y)}{W'^{(2t)}F(y)} + s[y]$;
for $y \in$ {message locations in GF$(q)$}
    if $(W^{(2t)} \neq 0)$ $errata[y] = 0$;
    else
        $errata[y] = \dfrac{-N^{(2t)}(y)}{W'^{(2t)}F(y)}$.
```

A candidate $W(z)$ is called ***legitimate*** if it has $|W|$ distinct roots among the code's unerased locations, and ***illegitimate*** otherwise. An illegitimate W might have roots among erased locations, or among elements of GF(q) which are not among the locations of a shortened RS code, or perhaps in some larger field, GF(q^m), which is an extension of GF(q). A legitimate W cannot have roots in any of these places, for that would violate the theorem of algebra which states that a polynomial can have no more roots than its degree.

The SCORING algorithm presented above ensures that every illegitimate W polynomial will receive a *score* $\geq$ BIG. By selecting the value of BIG to be sufficiently large, we can ensure that all illegitimate W's fail to compete successfully with some legitimate errata patterns, such as the syndrome, which corresponds to $N^{(0)} = 0$ and $W^{(0)} = 1$.

So the traditional method of scoring is now quite clear: For each $t = 0, 1, \cdots, r/2$, we can use the above algorithm to compute a value of SCORE[$2t$], corresponding to $N^{(2t)}/W^{(2t)}$. If this pair of polynomials represents a legitimate errata pattern, then SCORE[$2t$] is correct; otherwise, SCORE[$2t$] > BIG.

C. Generalization from $j = 2t$ to Arbitrary j

For traditional RS codes, for each odd value of j, $N^{(j)}/W^{(j)}$ does not correspond to any legitimate errata pattern, and so it need not be scored. However, these pairs of N and W have an interesting relationship to *doubly extended* RS codes, whose locations include not only the $q-1$ nonzero symbols in GF(q), but two additional locations: 0 and ∞. The parity-check matrix for such a code was introduced by J. Wolf [30]. Here is Wolf's matrix, with columns indexed by the code's locations:

D. Wolf's Parity-Check Matrix

$$
\begin{array}{cc}
\mathit{label} = & \begin{array}{cccccccc} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{-1} & 0 & \infty \end{array} \\
\mathcal{H} = & \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & \alpha & \alpha^2 & \alpha^3 & & \alpha^{-1} & 0 & 0 \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & & \alpha^{-2} & 0 & 0 \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & & \alpha^{-3} & 0 & 0 \\ & & & & \cdots & & & \\ 1 & \alpha^{r-1} & \alpha^{2r-2} & \alpha^{3r-3} & & \alpha^{1-r} & 0 & 1 \end{bmatrix}
\end{array}
$$

Here α is a primitive element of GF(q).

Any code vector may be conveniently written as

$$C = [C[1], C[\alpha], C[\alpha^2], \cdots, C[\alpha^{-1}], C[0], C[\infty]].$$

An examination of the $\mathcal{H}$ matrix reveals that the transpose of such a row vector lies in the nullspace of $\mathcal{H}$ if and only if

$$\sum_{y \in F} y^i C[y] = 0, \quad \text{for } i = 0, 1, 2, \cdots, r-2$$

and

$$C[\infty] + \sum_{y \in F} y^{r-1} C[y] = 0.$$

Here $\mathcal{F} = \text{GF}(q)$. These conditions are immediately seen to be equivalent to the conclusion of our Theorem 1, since in that theorem it is appropriate to take $C[y] = 0$ for all $y \in \mathcal{F} - \mathcal{S}$.

When j is odd, A3 and A5 yield

$$|W^{(j)}| \le \left\lfloor \frac{j}{2} \right\rfloor = \frac{j-1}{2} = |N^{(j)}|.$$

We should view this $N^{(j)}/W^{(j)}$ as a candidate solution with $\frac{i-1}{2}$ errors, $r-j$ erased check symbols at locations *ocheck*$[i]$, $j < i \le r$, and one erased message symbol located at ∞. If $|W^{(j)}| < \frac{i-1}{2}$, then $W^{(j)}$ will necessarily have fewer than $\frac{i-1}{2}$ roots among the unerased locations of the code, and we should view $W^{(j)}$ as illegitimate. The erased message symbol at location ∞ cannot be readable, because A5 asserts that $N^{(j)}$ is monic, and we have

$$C[\infty] = \frac{-N^{(j)}_{\lfloor \frac{j}{2} \rfloor}}{W^{(j)}_{\lfloor \frac{j}{2} \rfloor}}.$$

E. Modification to Algorithm of Appendix II

To score $N^{(j)}/W^{(j)}$ for arbitrary j, we need only replace $2t$ by j and modify the initialization to the following:

```
/*initialization*/
    SCORE = $\lfloor \frac{j}{2} \rfloor \cdot$ BIG;
    for ($i = j; i \le r; i++$){
        $y$ = rcheck[$i$];
        SCORE = SCORE + score[$y$];
    }
    if ($j$ is odd)
        SCORE = SCORE + score[$\infty$];
```

F. Addendum to Theorem 4

We must also append an additional case to the end of Theorem 4:

```
for $y \in$ {message location at $\infty$}
    if ($j$ is even) errata[$\infty$] = 0;
    else
        errata[$\infty$] = $\frac{N^{(j)}_{\lfloor \frac{j}{2} \rfloor}}{W^{(j)}_{\lfloor \frac{j}{2} \rfloor}}$;
```

IV. Algorithms with Worst Case Workload $\sim nr$

A. Global Scoring

The algorithm of Appendix III utilizes a scratchpad array of d registers which contain real numbers. The ith register in this scratchpad is used to accumulate the score of the candidate errata pattern corresponding to the solution with $s + 2t = i$, where i varies from 0 to $d-1$. This candidate corresponds to the presumption that there are $r - i$ erasured check symbols, located at *rcheck*[1], *rcheck*[2], $\cdots$, *rcheck*$[r-i]$. If i is odd, then we also have one erased message symbol, located at ∞. And we also have $\lfloor \frac{i}{2} \rfloor$ errors located at finite unerased symbols. The initialization corresponds to the conservative assumption that all erased symbols are unreadable, and that all error locations have the same reliability, called "BIG."

B. Erasure Matrix

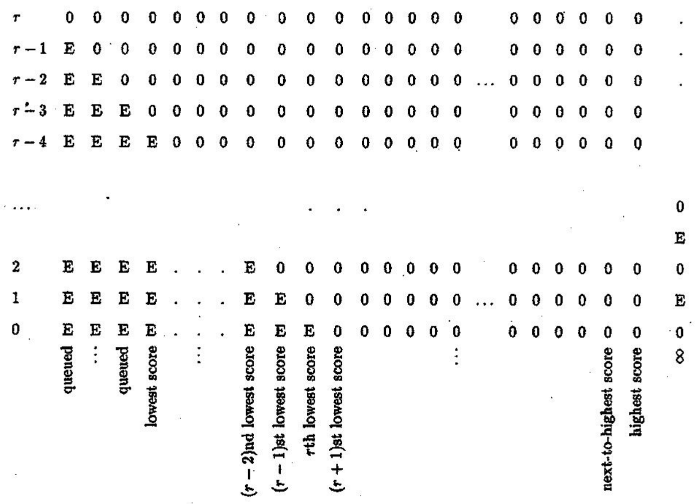
In the matrix at the top of the following page, columns represent locations and rows represent different values of i. The matrix entry indicates which symbols are viewed as erasures.

The SCORING algorithm must adjust the initialized values downward to account for readable erasures and for the actual error locations, each of which has a score less than BIG. For each finite unerased symbol, this entails examining $W^{(i)}(y)$ for all values of y. For each finite erased symbol, this entails examining both $W^{(i)}(y)$ and $N^{(i)}(y)$. Since there are n potential values of y and $d = r + 1$ potential values of i, the scoring algorithm requires access to about nd polynomial values. It is convenient to view the computation of these values as beyond the responsibility of the SCORING algorithm, which has access to the results of all such computations via appropriate $d \times n$ arrays, called $W[i, y]$ and $N[i, y]$. When appropriate, we also give the SCORING algorithm access to the results of computing $V^{(i)}(y)$ and $M^{(i)}(y)$ via similar arrays called $V[i, y]$ and $M[i, y]$.

In the prior subsection, we showed how to compute SCORE$[i]$ for any particular value of i. Although we might do that calculation for each value of i, in sequence, it is sometimes more advantageous to reverse the order of the two "for" loops, and to put the loop on i inside of the loop on y, as shown in Appendix III.

C. Scoring Algorithm for $s + 2t < d + 1$

If the true errata pattern has $s + 2t = d$, and if $i = d - 1 - 2t$, then according to Lemma 2, the correct message-error-locator,



$W(z)$, must be of the form

$$W(z) = -V^{(i)}(z) + cW^{(i)}(z)$$

where c is a scalar. If the received value in the next (erased) check location were readable, then the value of c would agree with the scalar $a[i]$ computed by the WB algorithm. But of course that corresponds to a degenerate case with smaller s, such that $s + 2t \leq d - 1$. In the nondegenerate case, the scalar c is unknown. But as detailed in Appendix IV, we can find all legitimate values of c by an appropriate search.

To this end, we replace our former one-dimensional array SCORE$[i]$ by a two-dimensional array, SCORE$[i, c]$. The index i runs from 0 through $d - 1$, as before, but the index c runs over all values in GF(q). For unshortened RS codes, $n = q - 1$ and $q = n + 1$. The one-dimensional SCORE array used in the previous subsection corresponds to the value with $c = \infty$. It also corresponds to letting i be one less and taking $c = a[i]$, and so there is no loss of generality in considering only finite choices of c.

We initialize all elements of the array SCORE$[i, c]$ to the same value as the initial value of our former SCORE$[i]$. As before, we then adjust for readable erasures, check errors, and message errors. Perhaps the simplest case is message errors, which we now explain. For each message location, y, we examine $W[i, y]$ and $V[i, y]$. But now, instead of testing whether $W[i, y] = 0$, we instead compute the quotient

$$c = V[i, y]/W[i, y].$$

This value of c is used as the index to make the appropriate adjustment to

$$\text{SCORE}[i, c].$$

For purposes of initialization, it is convenient to assume that the field size q is a power of 2 and that the elements of GF(q) can be represented as binary numbers as well as binary vectors. This assumption allows us to treat the second index of SCORE[,] as a natural integer for purposes of initialization, even though we know that for most purposes it is an element in a finite field.

The complete SCORING algorithm for $s + 2t < d + 1$ appears in Appendix IV.

After all adjustments corresponding to all n values of y have been completed, we then search the array of SCOREs to find the minimum value and the corresponding values of i and c.

Altogether, this SCORING algorithm splits into the following steps, of which only the first two are detailed in Appendix IV.

4a) Initialize the array SCORE$[i, c]$.
4b) For each code location, y, make d adjustments in this array.
4c) Search through this array to find the winning SCORE and the corresponding values of i and c.
4d) Using these values of i and c, determine the coefficients of the message-error-locator polynomial, and then find its roots by the old-fashioned method of exhaustive search.

Since the computational workload of each of these steps is overbounded by a term proportional to nd, so is the workload of the entire SCORING algorithm.

D. Reformulated WB *Decoding*

Traditional algebraic decoding algorithms [4], [28], [29] have all viewed the coefficients of the error-locator polynomial as a primary output of a routine which solved some appropriate set of key equations. However, the scoring or searching algorithms which follow this key equation solver need access to the *values* of the error-locator polynomial rather than to its coefficients. When there are many candidate error-locator polynomials, as is the case for bounded distance decoding, then the values of these polynomials at any particular point y must satisfy the same iterative relations that the polynomials satisfy when represented as power series in an indeterminate. We are thus led to view these iterative relations, represented by the scalar sequence of a's, as the primary output of the WB algorithm. Although the polynomials W, N, V, and M are still present, we can eradicate their coefficients entirely, and reformulate all computations in terms of their values $W[i,y]$, $N[i,y]$, $V[i,y]$, and $M[i,y]$, where y ranges over the erased checks. This reformulated algorithm can then be followed with additional computations which calculate $W[i,y]$ and $V[i,y]$ for all other values of i and y.

The algorithm which results from such a reformulation is given in Appendix V. For comparison, we have retained the same line numbers that appeared in Appendix I.

The worst case computational workload for this reformulated algorithm is proportional to nd. This same bound also applies to all other subprograms, except possibly the sorting program whose workload is proportional to $n \log n$ rather than to nd. But if d is as small as $\log n$, then the sorting program can be replaced by a series of d searches to find the minimum score, then the next-to minimum score, etc. So there is a uniform worst case upper bound on the computational workload of the "Bounded Distance + 1" RS Decoding algorithm which is proportional to nd.

Traditionally, the SCORING subprogram would follow this reformulated WB algorithm.

However, many other variations are possible, and the best choice may depend on other system considerations. We shall explore some of these possibilities in Section VI.

V. Performance and Tricks to Improve It

A. A High-Level View of SCORE's

There are some interesting bounds on the SCORE's of the candidate errata patterns which are independent of the values of the received symbols. These bounds depend only on the distribution of the scores.

Just as we SCORE errata patterns, we can also assign SCORE's to codewords. Indeed, it is conceivable that the actual errata pattern is a codeword, such as the all-zero codeword, and in that case the set of all possible errata patterns is the set of all possible codewords. Since there is a set of $(q-1)$ minimum weight RS codewords having nonzero values in any specified set of $d = r+1$ locations, it is clear that the minimum nonzero SCORE of all codewords may be found by summing the $r+1$ smallest *scores*. Let us call this parameter CMINSCORE. Let us also define some other terms, as follows:

CMINSCORE= Minimum SCORE among all codewords

ASCORE= SCORE of the actual errata pattern

EMINSCORE= Minimum SCORE among all errata patterns in the same coset as the received word

B-MINSCORE= Minimum SCORE among all errata patterns in some specified subset, $\mathcal{B}$

The appropriate choice of $\mathcal{B}$ is whatever set of errata patterns our decoding algorithm investigates. In the traditional case, this is all patterns which can be partitioned into t errors and s erasures including readable erasures, such that the s erasures are the s least reliable locations and for which

$$2t + s < d.$$

In Section IV-C, we have seen how our algorithm can weaken this condition to

$$2t + s < d + 1$$

Either of these choices of $\mathcal{B}$ must contain at least one candidate errata pattern whose SCORE is better than CMINSCORE, namely, the unique "erasures-only" errata pattern whose only nonzero values occur among the r symbols with lowest scores. In the pessimistic case, this SCORE might be as large as

$$\frac{r}{r+1} \cdot \text{CMINSCORE}.$$

So we have the following general bound:

$$\text{EMINSCORE} \le \text{B-MINSCORE} \le \frac{r}{r+1} \cdot \text{CMINSCORE}.$$

Furthermore,

If any candidate errata pattern has a

$$\text{SCORE} < \frac{\text{CMINSCORE}}{2}$$

then its SCORE is EMINSCORE.

For, if there were two such errata patterns, then their difference would have to be a codeword whose SCORE was less than CMINSCORE.

It is also clear, in general, that a theoretical maximum-likelihood decoding algorithm is one which finds an errata pattern with EMINSCORE. The max-likelihood decoder fails if

$$\text{ASCORE} > \text{EMINSCORE}$$

and only if

$$\text{ASCORE} \ge \text{EMINSCORE}.$$

B. The Parameters SCORE1 and SCORE2

More detailed results depend heavily on the particular parameters of the specific code and the channel [7], [21]. For the remainder of this section, I'll offer a generic summary of some of the relevant facts for some vaguely defined set of channels that I regard as typical.

One can specify parameters SCORE1 and SCORE2 such that

$$\frac{1}{2} < \frac{\text{SCORE1}}{\text{CMINSCORE}} < \frac{\text{SCORE2}}{\text{CMINSCORE}} < \frac{r}{r+1}$$

and

The probability that EMINSCORE > SCORE2 is negligibly small.

and

The probability that there are two errata patterns in the same coset with SCORE's < SCORE1 is negligibly small.

By "negligibly small," I mean comparable to the probability of max-likelihood decoding error.

One can estimate reasonable values for SCORE1 and SCORE2 by studying the distribution of SCORE's under the assumption that although the distribution of scores follows some appropriate model, the received symbols are distributed uniformly and independently at random, ignoring the constraints of the RS code. Then, with very high probability, there will be many candidates with SCORE < SCORE2, and no incorrect candidates with SCORE < SCORE1.

To fix ideas, we might imagine that SCORE1 = 70% CMINSCORE and SCORE2 = 80% CMINSCORE.

These results have several implications for our decoding algorithm.

First, in most situations, we should not accept any answer with SCORE > SCORE2. Even if SCORE corresponds to the best candidate in the set B, it is extremely likely to be incorrect. It is generally wiser to declare decoding failure than to accept such a weak candidate.

This observation does not necessarily disqualify the "all erasures" candidate. Rather, it imposes on that candidate, like others, the higher standard of a better-than-obvious SCORE. The "all erasures" candidate may have enough readable erasures to meet this standard.

Many of the winning low-score candidates which the max-likelihood decoder could find have many readable erasures, especially among the erased positions with highest scores, and many errors, especially among the unerased positions with lowest scores. If there is no big gap between consecutive ordered values of the scores of the received symbols, then the error rate among the unerased symbols with the lowest scores should be almost as high as the errata rate among the erased symbols with lowest scores. Among these positions with scores near the erasure threshold, one might plausibly argue that the fraction of readable erasures [6] and the fraction of errors, respectively, should each be nearly 50%. Most of the candidates in the set B, however, have only about $1/q$ or their erasures readable, and only about $t/(n-2t)$ of their unerased positions with low scores are in error.

In accordance with these results, it is reasonable for the portion of our algorithm which searches for the minimum SCORE among the errata patterns in B to maintain at least two thresholds: SCORE1 and SCORE2. Each new SCORE is compared with SCORE2, and the candidate is rejected unless SCORE < SCORE2, in which case the algorithm resets SCORE2 = SCORE and compares SCORE with SCORE1. If SCORE < SCORE1, the current candidate may be preemptively accepted as the winner, and the search can be terminated without examining any further candidates. If SCORE1 $\leq$ SCORE $\leq$ SCORE2, then the search continues with the reduced threshold of acceptability.

C. Using Location ∞

In Section III-C, we saw that two additional information symbols can be appended to RS codes to increase the length of the traditional cyclic code from $q-1$ to $q+1$, and that these additional locations are naturally associated with the symbols 0 and ∞. Although this has been known since the late 1960's [30], I am unaware of any application where it has ever been used.

Some code-preserving transformations have long been known [16], [19]. The canonical parity-check matrix of Section III-D is invariant under the affine group of permutations on GF (2^m). The generic permutation in this group is of the form

$$x \rightarrow ax + b$$

where x is the code location to be permuted, b is an arbitrary element of GF (2^m), and a is a nonzero element in GF (2^m). All such permutations have a fixed point at infinity. The subgroup of ROTATIONS, for which $b = 0$, also has a second fixed point at 0. The rotations are the well-known "cyclic shifts."

There is another interesting type of transformation which converts one RS code into another by multiplying each code symbol by an appropriate nonzero scalar. The symbols at locations ∞, zero, and one are multiplied by one, which leaves them invariant. The symbols at all other locations are multiplied by the $\frac{(r-1)}{2}$th powers of their locations. This converts the canonical RS parity-check matrix into the parity-check matrix for a noncanonical RS code which is *reversible.* If the symbols at 0 and ∞ are ignored, this code is cyclic and the reciprocal of every root of the generator polynomial is also a root of the generator polynomial. This property implies that the generator polynomial is a palindrome. This symmetry facilitates reductions of portions of the encoding hardware by much more than a factor of 2 [8]. Reversible RS codes are so attractive for encoding that they have been adopted as NASA standards, along with particular choices of bases to represent GF (2^m) over GF (2) in an especially advantageous manner [24]. The reversible form has also been used in many commercial applications.

When locations 0 and ∞ are included, the reversible RS code is readily seen to be invariant under the permutation called INVERSION. This permutation has a single fixed point at location "1." It swaps the symbols at locations 0 and ∞, and swaps every other location with its multiplicative inverse.

Of course, these transformations can be combined. We may start with the canonical code of Section III-D, transform it to reversible form, use the inversion permutation, and then transform the code back to the form of Section III-D. Combining these operations with the affine group described above, we conclude that the RS is invariant under a set of transformations, which includes all permutations in the linear fractional group

$$x \rightarrow \frac{ax-b}{cx-d}$$

where the determinant

$$\begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

is a nonzero scalar in GF (q). Each of these permutations must be accompanied by appropriate scalar multiplications of the permuted code symbols.

Since the linear fractional group is *triply transitive*, we might choose any ordered triplet of code locations and move them into any other. These symmetries imply that locations 0 and ∞ are as "natural" as any others.

In order to retain the advantages of the standard encoders, the form of the RS code appearing on the channel should be reversible. In order to use the decoding algorithms in Sections III-A and IV, the decoder should begin by transforming the received word to the canonical form of the RS code. After the canonical code has been decoded, it can be transformed back to the reversible code.

Even if symbols 0 and ∞ are absent from the code as it appears on the channel, it is advantageous to use them in the canonical form, and to exclude some other symbols instead. One interesting possibility is to pick the symbol with the $r/2$th least reliable score, and transform that symbol to location infinity. This then spreads the "erasures" over the least reliable $r+1$ symbols rather than only the least reliable r symbols. A review of the erasure matrix of Section IV-B reveals that all of the errata patterns meeting the former criteria are still considered, as well as many more. In the sense of Section V-A, the transformation to include ∞ as a "somewhat unreliable" location has doubled the size of $\mathcal{B}$, the candidate set of errata patterns. Many of the new candidates are plausible contenders.

D. Repeated Trials

If extra time is available, the decoder can attain further performance improvements by multiple passes through the WB algorithm. Each pass corresponds to a different ordering of the code's least reliable locations.

We have already discussed the most promising ordering, which corresponds to the classic recommendations of Forney [12]. One way to define additional orderings is as follows: For some given positive real value of ϵ, partition the lowest scores into some disjoint sets such that within any set, the maximum and minimum values differ by no more than ϵ. (This partition needs not be unique.) Then order the locations in the way which proceeds consecutively through each set in *reverse* order of scores, but proceeds from each set to the next in the usual increasing order.

The new orderings find many new candidate errata patterns that failed to make the first ordering by at most a small multiple of ϵ. The new orderings also again find some of the same candidates as the original ordering.

VI. Custom Engineering to Meet Design Goals

A. Minimizing Memory Requirements

It is possible to eliminate a considerable number of store-and-fetch instructions by integrating the WB algorithm and the scoring algorithm into a single entity. Most of the entries in the W, N, V, and M arrays are then used only immediately after their values are computed. In many architectures, these values can be stored only in fast temporary registers, and most of the memory requirements of these arrays can be eliminated.

However, we shall see that this decision may entail tradeoffs against other design goals.

B. Average-Case Workloads

Many communication or computer memory systems require continuous throughput, but a latency of several RS code blocks is often acceptable. In such circumstances, the received code blocks can be fed into a buffer, or queue. The successive codewords are extracted from this queue and processed by the decoder on a first-in, first-out basis. Even though the decoder must keep up with the channel, it may be permitted to fall behind occasionally as long as buffer overflow is a very rare and recoverable event. If the average speed of the decoder is considerably faster than the channel, and if the distribution of its running times is sufficiently good, then the probability of buffer overflow may be even smaller than the probability of max-likelihood decoding error.

Similar strategies were used by sequential decoders in the early 1960's [11], [31]. They fell into disfavor because the distribution of decoding times for sequential decoders had Paretian tails [14]. Some of the moments of such distributions are infinite. Although the probability of decoding error was *very* small, the probability of overflowing even a very large buffer turned out to be unacceptably large.

On the other hand, the decoding times for Reed–Solomon codes typically have distributions with tails that approach zero very rapidly. In some interesting situations, the buffered approach can gain more than an order of magnitude in speed, while perhaps increasing the probability of decoding error from 10^{-12} to only 2×10^{-12}. Furthermore, in some disk-memory applications, there need be no degradation in decoded data integrity at all, and the speedup can be accomplished with a buffer of only 4 or 5 code blocks. This is because it may be possible to replace the "buffer overflows" by "pauses," or "slipped revolutions" during which no data is read. A "pause" rate of 1% is generally considered to be very high; it is an easy design target which can often be achieved by relatively rudimentary considerations. Yet a 1% pause rate is far preferable to an additional 1% redundancy. The 1% redundancy affects *both* throughput and total storage capacity. The 1% pause rate has no effect on total storage capacity, and its effect on throughput is even less than 1%, because some of the pauses are hidden by latency which occurs when the reading jumps between different sectors.

So it is often desirable to reduce the *average* decoding time. Only a slightly different, more sophisticated software design goal is to reduce the tilted average [13], [14] decoding time, where the tilt is chosen so the result closely approximates the goal of minimizing the probability of buffer overflow.

C. Replace Buffer Overflows by Punts

One way to ensure that the buffer *never* overflows is to allow the decoder to *punt* whenever it is sufficiently far behind. The "punt" allows one or more blocks to leave the buffer undecoded. The "punt" feature can be embedded in carefully written software, or in special-purpose hardware.

D. Replace Initializations with Finalizations

In some conventional software programs, not necessarily concerned with decoding, initializations consume an unduly large fraction of the average workload. This situation can be improved by adopting the strategy used at fire stations: "Don't initialize, finalize!" When a task has just been completed, it is still evident what has been used and what has not. Only those things which have been used need be restored before work on the next task can begin. Finalization avoids unnecessary reinitializations of many things which were not used.

E. Best Case Performance: The Optimist's View

An important strategy to minimize average decoding time is called *best case* optimization. This means that the software designer's first goal is to minimize the decoding time *when there are no errata.* After that has been accomplished, he then pursues the secondary goal of minimizing the decoding time when there is *only one erratum.* He then minimizes the decoding time when there are *only two errata*, etc. Heuristically, the reason this works so well is that RS codes are often designed to yield very high reliability, with undecodable rates, per block, of only 10^{-12} or better. If the code can correct up to e errata, and the probability of more than e errata is that small, then the probability of exactly e errata is likely to be only one or two orders of magnitude larger, and the probability of $e-1$ errata may be only one or two orders of magnitude larger than that, etc. Not surprisingly, the probability of *no errata* in the entire block is often significant, sometimes even greater than 50%.

Philosophically, a best case decoder is analogous to a small child who continually asks, "Are we almost there now?" This question may occur at many places in a long decoding program. But, in a high-reliability application, the odds are quite favorable that any time the question is asked, the answer is likely to be "YES."

F. Erasures Only

In the highest level of the bounded distance decoding algorithm featured in this paper, best case optimization should begin within the "erasures-only" decoding routine, which begins with the original errata pattern in which all of the errata are located at the r check locations specified by the hardware of the re-encoder. The erasures-only algorithm ends with the syndrome $s[y_i]$, all of whose errata are located at the r locations with the least reliable scores. A best case erasures-only decoding algorithm will migrate between these two syndromes via an iteration that proceeds one location at a time. The ith step is to find an errata pattern which is confined to the re-encoding hardware's i least reliable message locations and its $(r-i)$ least reliable check locations. The difference between the syndromes corresponding to any two consecutive values of i is an RS codeword of minimum weight $d=r+1$. Except for a single scalar N_0, the values of the coefficients of this codeword are given by Theorem 1, and the value of this scalar must be chosen to ensure that the next syndrome has zero errata value in the location that is forbidden at the next level of the iteration. While the best case-optimized erasures-only algorithm computes the syndrome corresponding to the next value of i, it also computes its score. It then repeats the question: "Is this score sufficiently small?" Section V-B offers a quantitative notion of "sufficiently small," namely, SCORE1 and SCORE2. If all of the errata are confined to the code's least reliable i positions, then many of the errata values in the newly computed syndrome will be zero, and the answer will be "YES."

This erasures-only decoding algorithm based on the best case optimization gives a worst case running time which is essentially as good as any other known erasures-only decoding algorithm. So, in some sense, it costs virtually nothing to incorporate the best case philosophy into an erasures-only decoding algorithm.

G. WB and SCORING

If some errata occur within the code's $n-r$ most reliable locations, then the erasures-only decoding algorithm runs to completion without ever obtaining an affirmative answer to its "Are we almost done now?" question. The WB algorithm then begins. Each time the value of j is incremented, a dogmatically best case optimized algorithm will proceed in the way that minimizes the remaining workload under the presumption that the current value of j corresponds to some errata pattern with very low score. This algorithm would complete the computations of SCORE$[j,c]$ for all values of c before incrementing the next value of j. If any of these values is sufficiently low, then it can be declared the winner and both WB and SCORING may be terminated.

There is also an important best case suboptimization within the portion of the algorithm which calculates those portions of the SCORE's corresponding to message errors. It is best to process the message locations in order of increasing reliability. The locations with lower scores are more likely to correspond to errors, and there is some chance that the program may be able to find the roots of W and then terminate the search because SCORE $<$ SCORE1.

Incorporating the best case philosophy into the WB and SCORING algorithms may entail some nontrivial costs. The values of $W[j,y]$ and $V[j,y]$ must be stored for all values of y. This entails not only the use of $2q$ additional memory locations, it also consumes time spent on additional store-and-fetch instructions. For this reason, on some architectures, the variation of the integrated WB and SCORING algorithms which embody the best case philosophy may have worst case workloads which are degraded by nearly a factor of two from workloads of algorithms which eschew best case optimization.

H. The Pessimist's View

The pessimist believes that the decoder will fail to find the correct answer. In particular, he expects every SCORE to remain above SCORE2. Like the evil landlord of the classic melodrama, he is forever looking for reasons to declare default. If the decoder is indeed destined to fail, then the sophisticated pessimist may be able to save considerable time by finding a convincing proof of the inevitable failure much faster than by the simple-minded completion of the search. More importantly, by terminating doomed portions of the search more quickly, the pessimistic approach may also expedite the completion of successful searches.

There are several ways in which the pessimistic philosophy can be used to speed up SCORING.

High-rate codes have many more information symbols than check symbols, and so the SCORING program's primary workload involves searching for roots of candidate W's. We now assume that $W[j, y]$ (and/or the comparable expression $-V[j, y] + cW[j, y]$) is SCOREd via an outer loop in which y runs over the message locations in order of increasing score, and an inner loop in which j runs from 0 to *jmax*.

The SCORING algorithm can create and maintain a value of SCORE3, the best partial SCORE it has actually yet seen, even if that value exceeds SCORE2. It can also maintain the next-to-best score yet seen, SCORE4. And it can maintain similar values for each row, SCORE3[j] and SCORE4[j]. At strategic breakpoints in the search, the program can examine SCORE3[j]. The pessimist hopes that this value is so big that even if the needed roots are all found consecutively and immediately, the resulting score will still be higher than SCORE2. If this pessimistic conclusion is true, then the value of *jmax* can be decreased, and the workload savings may be substantial.

I. Optimism and Pessimism Combined!

Whenever SCORE3 or SCORE3[j] is decreased, the optimist may seek to interrupt the proceedings to force a detailed investigation of the relevant candidate. The coefficients of the candidate polynomial, W, can be computed, and all linear factors corresponding to roots that have already been found can be factored out. Then, if the residual W polynomial has sufficiently small degree, its roots can be found by algebraic methods [4], [5]. These methods are much faster than exhaustive search, especially in the trivial case when $|W| = 1$ and in the quadratic in which $|W| = 2$. The quartic case $|W| \leq 4$ is also attractive in some hardware implementations, and even larger degrees become competitive if the code length is sufficiently large. If, as the pessimist expects, the residual polynomial has fewer roots than its degree, these methods will yield that answer.

There are circumstances in which the optimist and the pessimist are *both* eager to interrupt the general searching procedure of the SCORING algorithm in order to pursue a detailed investigation of a particular candidate. The optimist will favor this action for every candidate that reduces its SCORE3[j]. One set of conditions sufficient for the pessimist to concur is when SCORE4[j], SCORE3[$j-1$], and perhaps SCORE3[$j-2$] are all much larger than SCORE3[j], and when y is among the first message locations to be examined. In those circumstances, rejection of the current candidate would enable a big reduction of *jmax*, which will yield a large reduction in the remaining workload.

J. Tradeoffs Between Speed and Performance

The pessimist we have been considering is an ideological purist. He seeks a complete and rigorous proof that all SCORE's > SCORE2.

We can expedite his workload substantially by weakening his goal, and requiring him only to investigate the SCORE's of candidates that meet some additional criteria. This necessarily entails some loss of performance, because there is some chance that one of the disqualified candidates might have been a winner. But in many cases it is possible to select criteria such that this performance loss is much less than the performance gained by searching to $d+1$ instead of only to d.

As expounded elsewhere [7], genuine errata patterns tend to conform to the statistics of the channel noise environment, whereas spurious errata patterns tend instead to conform to the rather different statistics imposed by artifacts of the code and its decoding algorithm. In the case of the present paper, the channel noise environment is represented by the raw symbol scores, *score*[y]. The artifacts of the decoding algorithm arise from the partition of each set of errata into s erasures and t errors. Only $\frac{1}{q}$ of the erasures of spurious errata patterns are readable [6], and this fraction is significantly less than the number of readable erasures of genuine errata pattern. Similarly, errors in genuine errata patterns are heavily biased toward locations with low scores; errors in spurious errata patterns are distributed uniformly.

As a specific example, let the code be one of the NASA standards, with $n = 255$, $r = 16$ or 32. As a baseline to inspire further work, we propose a specific *ad hoc* criterion. The reader is invited not only to tinker with our *ad hoc* parameter values, but to devise other criteria that might be more effective than the following:

Define a set of ten "transitional" locations as follows:
For $j \geq r-5$, the ten transitional locations are the ten least reliable received symbols.
For $j \leq r-5$, the ten transitional locations are the five most reliable erased symbols and the five least reliable unerased symbols.
Define an erased location to be "confirming" if it satisfies the equations for a readable erasure.
Define an unerased location to be "confirming" if it is a root of $W(z)$.

Then impose the primary criterion that disqualifies any errata pattern unless either $(2t + s < d)$ or the candidate has at least three confirming locations among its ten transitional locations.

We also impose the secondary criteria that, for all $i \leq 11$, a candidate have no more than $i+3$ errors among the most reliable $20i$ locations.

Most genuine candidates satisfy these criteria, but most spurious ones are eliminated. The criteria also reduce computational workload. The reduction is often significantly more than a factor of 2 in speed, but it is probabilistic. We might set a limit on the total decoding time, and declare a decoding error whenever the limit is not met. This imposes an additional potential cause of decoding error, called "timeout." Ideally, the "timeout" would be monitored as an average figure computed over several consecutive blocks. If appropriate to the system requirements, one could also compute averages on a very strict single-block basis.

Although this workload is measured in an average sense, this average is different from the optimistic averages discussed in Sections VI-B and VI-E. Those averages depended on "typical" channel behavior, which implied a very high probability of errata patterns significantly less severe than the algebraic capabilities of the code. The present averaging is needed primarily to get over the ensemble of spurious errata patterns. It should yield acceptable timeouts even under channel conditions which are so severely degraded that the probability of decoding error remains abnormally high over a long sequence of consecutive blocks.

By considering only candidates which meet the specified criteria, we can reduce the total time limit, from its "worst case" value proportional to rn, to a lower value more nearly proportional to r^2, while the degradations due to timeouts remain inconsequential.

To achieve this goal, we must eliminate all arrays of sizes proportional to nr, including SCORE$[j, c]$. To this end, for each j, we simply list and sort the ten values of c corresponding to the transitional locations. The listed c's are then sorted and compared to find replications. Candidates with insufficient replications are disqualified. In the unlikely event that there is more than one surviving candidate, the current preliminary SCORE's are compared and used to eliminate all but perhaps the top two contenders, which are then examined sequentially. The candidate with the lower value of j is examined first. Using its value of c, the coefficients of its error-locator polynomial is determined, and a search for roots begins. If roots fail to appear sufficiently quickly to meet the secondary criteria, then the search can be aborted. So the probability of spending much time on spurious candidates is quite small. A significant fraction of the computational workload is consumed on the final stages of scoring the genuine winning candidate, which is nearly always unique.

Philosophically, error-pattern searches have much in common with executive searches.

APPENDIX I "ERRORS-ONLY" WELCH–BERLEKAMP ALGORITHM, WITH QUEUE

Input Declarations:

r is the redundancy of the code, a constant integer.

$F(z)$ is a monic polynomial of degree r over GF(q). Its roots are the check locations.

$F'(z)$ is the formal derivative of $F(z)$.

$s[]$ is an array of dimension r. Its yth element is the difference between the received value of a check and the value of that check obtained by re-encoding the received message symbols. These are elements in GF(q).

ocheck[] is an array of dimension r. Its elements are the check locations, in their original order. They are elements in $GF(q)$.

Other Declarations:

rcheck[] is an array of dimension r. The algorithm will reorder the check locations and write the reordered check locations into this array.

yqueue[] is another initially empty array, of maximum dimension r.

$a[]$ is an array of dimension $r+1$ over GF(q), whose elements we call $a_0, a_1, a_2, \cdots$.

$j, k, qin, qout$ are natural integers used as indices.

$N^{(j)}, M^{(j)}, V^{(j)}, W^{(j)}$ are all polynomials in z, with coefficients in GF(q).

Initializations:

```
        $N^{(0)} = 0; M^{(0)} = V^{(0)} = W^{(0)} = 1;$
        $j = qin = qout = k = 0;$
  1:  for ($k = 0,\ k < r;\ k++$){
  2:     ASSERTIONS;
  3:     $y = ocheck[k];$
  4:     $b = N^{(j)}(y) - W^{(j)}(y)F'(y)s[y];$
  5:     if ($b == 0$)$yqueue[++qin] = y;$
  6:     else {
  7:        $c = M^{(j)}(y) - V^{(j)}(y)F'(y)s[y];$
  8:        $a_j = \frac{c}{b};$
  9:        $rcheck[j] = y;$
*10:        $W^{(j+1)} = -V^{(j)} + a_j W^{(j)};$
*11:        $N^{(j+1)} = -M^{(j)} + a_j N^{(j)};$
*12:        $V^{(j+1)} = (z - y)W^{(j)};$
*13:        $M^{(j+1)} = (z - y)N^{(j)};$
 14:        $++j;$
 15:        if ($qin > qout$){
 16:           $y = yqueue[++qout];$
 17:           $b = N^{(j)}(y) - W^{(j)}(y)F'(y)s[y];$
 18:           assert ($b \neq 0$);
 19:           $c = M^{(j)}(y) - V^{(j)}(y)F'(y)s[y];$
 20:           assert ($c == 0$);
 21:           $a_j = 0;$
 22:           $rcheck[j] = y;$
*23:           $W^{(j+1)} = -V^{(j)};$
*24:           $N^{(j+1)} = -M^{(j)};$
*25:           $V^{(j+1)} = (z - y)W^{(j)};$
*26:           $M^{(j+1)} = (z - y)N^{(j)};$
 27:           $++j;$
 28:        }
 29:     }
 30:     ASSERTIONS;
      }
```

APPENDIX II SCORE $N^{(2t)}/W^{(2t)}$

```
/*declaration*/
BIG
   is a constant large positive real number, which exceeds
   any acceptable final value of SCORE.
/*initialization*/
   SCORE = $|W^{(2t)}| \cdot$ BIG;
   for ($i = 2t; i \leq r; i++$){
      $y = rcheck[i];$
      SCORE = SCORE + $score[y];$
   }
/*computation*/
   /*readable erasures*/
      for ($i = 2t; i \leq r; i++$){
        $y = rcheck[i]$;
        if $(N^{(2t)}(y) == W^{(2t)}(y)F'(y)s[y])$
            SCORE = SCORE − $score[y]$;
    }
/*erroneous checks*/
    for $(i = 2t - 1; i \geq 0; i - -)\{$
        $y = rcheck[i]$;
        if $(W^{(2t)}(y) == 0)$
            SCORE = SCORE + $score[y]$ − BIG;
    }
/*erroneous messages*/
    for $(i = 0; i < n - r; i + +)\{$
        $y = message[i]$;
        if $(W^{(2t)}(y) == 0)$
            SCORE = SCORE + $score[y]$ − BIG;
    }
```

APPENDIX III
GLOBAL SCORING FOR $s + 2t < d$

```
/*initialization, assuming $r$ is even*/
    SCORE$[r] = \frac{r}{2} \cdot$ BIG
    for $(i = r - 2; i \geq 0; i = i - 2)$ {
        SCORE$[i{+}1]$ = SCORE$[i{+}2]$ + $score[rcheck[i]]$ + $score[\infty]$ − BIG
        SCORE$[i]$ = SCORE$[i{+}1]$ + $score[rcheck[i]]$ − $score[\infty]$
        }
for $(k = 1; k \leq r; k + +)\{$
    $y = rcheck[k]$;
    $Fp = F'(y)$;
    for $(i = 0; i < k; i + +)$
/* readable erasure */
    if $(N[i,y] == W[i,y] \cdot Fp \cdot s[y])$
        SCORE$[i]$ = SCORE$[i]$ − $score[y]$;
    for $(i = k; i \leq r; i + +)$
/* error at check location*/
    if $(W[i,y] == 0)$
        SCORE$[i]$ = SCORE$[i]$ + $score[y]$ − BIG;
}
for $(k = 1; k <= n - r; k + +)\{$
    $y = message\,[k]$;
    for $(i = 0; i \leq r; i + +)$
/* error at message location*/
    if $(W[i,y] == 0)$
        SCORE$[i]$ = SCORE$[i]$ + $score[y]$ − BIG;
}
```

APPENDIX IV
GLOBAL SCORING FOR $s + 2t < d + 1$

```
/* initialization */
for $(c = 0; c < q; c + +)$
SCORE$[r, c]$ = 0;
for $(i = r - 1; i \geq 0; i - -)\{$
    for $(c = 0; c < q; c + +)$
        SCORE$[i, c]$ = SCORE$[i{+}1, c]$ + $score\,[rcheck\,[\,i\,]]$ ; }
}
 for $(i = 1; i \leq r; i = i + 2)\{$
    for $(c = 0; c < q; c + +)$
        SCORE$[i, c]$ = SCORE$[i{+}1, c]$ + $score[\infty]$;
}
 for $(i = 0; i \leq r; i + +)\{$
    for $(c = 0; c < q; c + +)$
        SCORE$[i, c]$ = SCORE$[i, c]$ + BIG $\cdot \left\lceil \frac{i+1}{2} \right\rceil$
}
/* SCORING */
for $(k = 1; k \leq r; k + +)\{$
    $y = rcheck\,[\,k - 1\,]$ ;
    $Fp = F'(y)$;
    for $(i = 0; i < k; i + +)\{$
/* readable erasure */
        $c = \dfrac{(M[i,y] - V[i,y] \cdot Fp \cdot s[v])}{(N[i,y] - W[i,y] \cdot Fp \cdot s[v])}$
        SCORE$[i, c]$ = SCORE$[i, c]$ − $score[y]$;
    }}
    for $(i = k; i <\leq r; i + +)\{$
/* error at check location*/
        $c = \dfrac{V[i,y]}{W[i,y]}$
        SCORE$[i, c]$ = SCORE$[i, c]$ + $score[y]$ − BIG;
    }
}}
for $(k = 1; k \leq n - r; k + +)\{$
    $y = message\,[k]$;
    for $(i = 0; i \leq r; i + +)\{$
/* error at message location*/
        $c = \dfrac{V[i,y]}{W[i,y]}$
        SCORE$[i, c]$ = SCORE$[i, c]$ + $score[y]$ - BIG ;
    }
} }
```

APPENDIX V
REFORMULATED WELCH–BERLEKAMP ALGORITHM, TO COMPUTE $N[i,y]$ AND $W[i,y]$

```
DECLARATIONS;
$j = qin = qout = k = 0$;
 1: for $(k = 0,\ k < r;\ k + +)\{$
 3:     $y = ocheck[k]$;
        $N[0,y] = 0; M[0,y] = V[0,y] = W[0,y] = 1$;
        for $(i = 0; i < j; i + +)\{$
            $W[i + 1, y] = -V[i, y] + a_i W[i, y]$;
            $N[i + 1, y] = -M[i, y] + a_i N[i, y]$;
            $V[i + 1, y] = (y - rcheck[i])W[i, y]$;
            $M[i + 1, y] = (y - rcheck[i])N[i, y]$;
```

```
      }
4:    $b = N[j,y] - W[j,y]F'(y)s[y]$;
5:    if $(b == 0)\{yqueue[++qin] = y; jqueue[qin] = j;\}$
6:    else {
7:      $c = M[j,y] - V[j,y]F'(y)s[y]$;
8:      $a_j = \frac{c}{b}$;
9:      rcheck [j] = y ;
14:     $++j$;
15:     if $(qin > qout)\{$
16:       $y = yqueue[++qout]$;
          for $(i = jqueue[qout]; i < j; i++)\{$
            $W[i+1,y] = -V[i,y] + a_iW[i,y]$;
            $N[i+1,y] = -M[i,y] + a_iN[i,y]$;
            $V[i+1,y] = (y - rcheck[i])W[i,y]$;
            $M[i+1,y] = (y - rcheck[i])N[i,y]$;
21:         $a_j = 0$;
22:         $rcheck[j] = y$;
27:         $++j$;
28:       }
29:   }
    }
/* Complete Evaluations of $W$ and $M$ for Queued Check Locations */
  for $(k = qout; k < qin; k++)\{$
      $y = queue[k]$;
*   ← rcheck [j + k - qout] = y ;
      for $(i = 0; i < j; i++)\{$
          $W[i+1,y] = -V[i,y] + a_iW[i,y]$;
          $N[i+1,y = -M[i,y] + a_iN[i,y]$;
          $V[i+1,y] = (y - rcheck[i])W[i,y]$;
          $M[i+1,y] = (y - rcheck[i])N[i,y]$;
      }
  }
  /* Evaluate $W$ for Message Locations Among $\{rchecks\}$ */
  for $(k = 0; k < j; k++)\{$
      $y = rcheck[k]$;
      for $(i = k; i < j; i++)\{$
          $W[i+1,y] = -V[i,y] + a_iW[i,y]$;
          $V[i+1,y] = (y - rcheck[i])W[i,y]$;
      }
  }
  /* Evaluate $W$ for All Other Message Locations */
  for $(k = 0; k < n - r; k++)\{$
      $y = message[k]$;
      $V[0,y] = W[0,y] = 1$;
      for $(i = 0; i < j; i++)\{$
          $W[i+1,y] = -V[i,y] + a_iW[i,y]$;
          $V[i+1,y] = (y - rcheck[i])W[i,y]$;
      }
  }
```

*Next line is necessary only to **set up** rcheck for scoring algorithm.

References

[1] K. Araki, M. Takada, and M. Morii, "The generalized syndrome polynomial and its application to the efficient decoding of Reed-Solomon codes based on GMD criteria," presented at the 1993 Int. Symp. on Information Theory, San Antonio, TX.

[2] G. A. Baker, Jr. and P. R. Graves-Morris, *Padé Approximants, Part I: Basic Theory*, vol. 13 of G-C Rota, Ed., *Encyclopedia of Mathematics and Its Applications*. Reading, MA: Addison-Wesley, 1981.

[3] ——, *Padé Approximants, Part II: Extensions and Applications*, vol. 14 of G-C Rota, Ed., *Encyclopedia of Mathematics and Its Applications*. Reading MA: Addison-Wesley, 1981.

[4] E. R. Berlekamp, *Algebraic Coding Theory*. New York: McGraw-Hill, 1968 and Laguna Hills, CA: Aegean Park Press, 1984.

[5] ——, "Factoring polynomials over large finite fields," *Math. Computat.*, vol. 24, no. 111, pp. 713–735, July 1970.

[6] E. R. Berlekamp and J. L. Ramsey, "Readable erasures improve the performance of Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. IT-24, pp. 384–386, 1978.

[7] E. R. Berlekamp, "The technology of error-correcting codes," *Proc. IEEE*, vol. 68, pp. 564–593, 1980.

[8] ——, "Bit-serial Reed-Solomon encoders," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 869–874, 1982.

[9] E. R. Berlekamp, G. Seroussi, and P. Tong, "Hypersystolic Reed-Solomon decoder," U. S. Patent 4 958 348, issued Sept. 18, 1990.

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

[11] R. M. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Trans. Inform. Theory*, vol. IT-9, pp. 64–74, 1963.

[12] G. D. Forney, "Generalized minimum distance decoding," *IEEE Trans. Inform. Theory*, vol. IT-12, pp. 125–131, 1966.

[13] R. G. Gallager, *Information Theory and Reliable Communication*. New York: Wiley, 1968.

[14] I. M. Jacobs and E. R. Berlekamp, "A lower bound to the distribution of computation for sequential decoding," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 167–174, 1967.

[15] J. Justesen, "On the complexity of decoding Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. IT-22, pp. 237–238, 1976.

[16] T. Kasami, S. Lin, and W. W. Peterson, "Some results on cyclic codes which are invariant under the affine group and their applications," *Inform. Contr.*, vol. 11, pp. 475–496, 1967.

[17] D. E. Knuth, *The Art of Computer Programming—Sorting and Searching, Vol 3*. Reading, MA: Addison-Wesley, 1973.

[18] R. Kotter, "A new efficient error-erasure location scheme in GMD decoding," presented at the 1993 Int. Symp. on Information Theeory, San Antonio, TX.

[19] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, vols. 1 and 2. Amsterdam, The Netherlands: North-Holland, 1977.

[20] J. L. Massey, "Shift register synthesis and BCH decoding," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 122–127, 1969.

[21] R. J. McEliece, *The Theory of Information and Coding*, vol 3 of G-C Rota, Ed., *Encyclopedia of Mathematics and Its Applications*. Reading MA: Addison-Wesley, 1981.

[22] M. Morii and M. Kasahara, "Generalized key-equation of Remainder Decoding Algorithm for Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. 38, pp. 1801–1807, 1992.

[23] N. J. Patterson, "Algebraic decoding of Goppa codes," *IEEE Trans. Inform. Theory*, vol. IT-21, pp. 203–207, Mar. 1975.

[24] M. Perlman, "A comparison of conventional Reed-Solomon encoders and Berlekamp's architecture," JPL Tech. Brief 5240, NASA Patent Office D15568.

[25] W. W. Peterson, *Error-Correcting Codes*. ambridge, MA: MIT Press, 1961.

[26] D. Slepian, "A class of binary signaling alphabets" *Bell Syst. Tech. J.*, vol. 35, pp. 203–234, 1956.

[27] U. K. Sorger, "A new Reed-Solomon code decoding algorithm," *IEEE Trans. Inform. Theory*, vol. 39, pp. 358–365, Mar. 1993.

[28] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equation for decoding Goppa codes," *Inform. Contr.*, vol. 27, pp. 87–99, 1975.

[29] L. R. Welch and E. R. Berlekamp, "Error correction for algebraic block codes," U. S. Patent 4 633 470, issued Dec. 30, 1986.

[30] J. K. Wolf, "Adding two information symbols to certain nonbinary BCH codes and some applications," *Bell Syst. Tech J*, vol. 48, no. 7, pp. 2405–2424, Sept. 1969.

[31] J. M. Wozencraft and I. M. Jacobs, *Sequential Decoding*. Cambridge MA: MIT Press, 1961.