We divide it into three cases.

1) $n_{N,2} = 0, 1$
   If $n_{F,2} \geq 0$, then $n_{N,2} \leq 2 + n_{F,2}$ holds. Thus,
   $$2^{n_{N,2}} | (2^2 \cdot 2^{n_{F,2}} \cdot 2^{n_{G,2}}).$$

2) $n_{N,2} = 2, 3, 4$
   If $n_{F,2} \geq 1$, then as required by Lemma 3.3, $n_{G,2} \geq 1$. Thus, $n_{N,2} \leq 2 + n_{F,2} + n_{G,2}$ holds and consequently
   $$2^{n_{N,2}} | (2^2 \cdot 2^{n_{F,2}} \cdot 2^{n_{G,2}}).$$

3) $n_{N,2} \geq 5$
   If $n_{N,2} \geq 5$, then
   $$\left\lceil \frac{n_{N,2} - 2}{2} \right\rceil > 1.$$
   By Lemma 3.3, if $n_{N,2} - 1 > n_{F,2} \geq \left\lceil \frac{n_{N,2}-2}{2} \right\rceil$, then $n_{G,2} = n_{F,2}$. Consequently, if $n_{N,2}$ is even
   $$2 + n_{F,2} + n_{G,2} = 2 + 2 \cdot n_{F,2} \geq 2 + n_{N,2} - 2 = n_{N,2}$$
   and if $n_{N,2}$ is odd
   $$2 + n_{F,2} + n_{G,2} = 2 + 2 \cdot n_{F,2} \geq 2 + n_{N,2} - 1 > n_{N,2}.$$
   Thus, $2^{n_{N,2}} | (2^2 \cdot 2^{n_{F,2}} \cdot 2^{n_{G,2}})$. If $n_{F,2} \geq n_{N,2} - 1$, by Lemma 3.3, $n_{G,2} \geq n_{N,2} - 1$. Thus, $2 + n_{F,2} + n_{G,2} \geq 2 \cdot n_{N,2} > n_{N,2}$ and consequently $2^{n_{N,2}} | (2^2 \cdot 2^{n_{F,2}} \cdot 2^{n_{G,2}})$. □

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers whose comments and suggestions have greatly helped in improving the quality of the original manuscript.

REFERENCES

[1] J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 101–119, Jan. 2005.
[2] O. Y. Takeshita and D. J. Costello Jr., "New deterministic interleaver designs for turbo codes," *IEEE Trans. Inf. Theory*, vol. 46, no. 6, pp. 1988–2006, Sep. 2000.
[3] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 1249–1253, Mar. 2006.
[4] S. Dolinar and D. Divsalar, "Weight distribution for turbo codes using random and nonrandom permutations," TDA, Progr. Rep. 42–122, Aug. 1995.
[5] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc. Int. Conf. Communications*, Seattle, WA, Jun. 1995, pp. 54–59.
[6] S. Crozier and P. Guinand, "High-performance low-memory interleaver banks for turbo-codes," in *Proc. 54th IEEE Vehicular Technology Conf. (VTC 2001 Fall)*, Atlantic City, NJ, Oct. 2001, pp. 2394–2398.
[7] S. Crozier and P. Guinand, "Distance upper bounds and true minimum distance results for turbo-codes designed with DRP interleavers," in *Proc. 3rd Int. Symp. Turbo Codes and Related Topics*, Brest, France, Sep. 2003, pp. 169–172. Ecole Nationale Supérieure des Télécommunications de Bretagne .
[8] F. Daneshgaran and M. Mondin, "Design of interleavers for turbo codes: Iterative interleaver growth algorithms of polynomial complexity," *IEEE Trans. Inf. Theory*, vol. 45, no. 6, pp. 1845–1859, Sep. 1999.
[9] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding : Turbo-codes," in *Proc. Int. Conf. Communications*, Geneva, Switzerland, May 1993, pp. 1064–1070.
[10] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, and G. Nebe, "Interleaver design for turbo codes," *IEEE J. Sel. Areas Commun.*, vol. 19, no. 5, pp. 831–837, May 2001.
[11] C. J. Corrada-Bravo and I. Rubio, "Deterministic interleavers for turbo codes with random-like performance and simple implementation," in *Proc. 3rd Int. Symp. Turbo Codes*, Brest, France, Sep. 2003.
[12] B. Moision and M. Klimesh, "Some observations on permutation polynomials," JPL, Inter-office Memo. 331.2005.1.1.
[13] M. Cheng, M. Nakashima, J. Hamkins, B. Moision, and M. Barsoum, "A decoder architecture for high-speed free-space laser communications," *Proc. SPIE*, vol. 5712, pp. 174–185, Apr. 2005.
[14] J. Gathen and J. Gerhard, *Modern Computer Algebra*, 1st ed, U.K.: Cambridge Univ. Press, 1999.
[15] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, 5th ed. Oxford, U.K.: Oxford Univ. Press, 1979.
[16] R. L. Rivest, "Permutation polynomials modulo 2^w," *Finite Fields Their Applic.*, vol. 7, pp. 287–292, 2001.
[17] G. Mullen and H. Stevens, "Polynomial functions $(\mathrm{mod}\ m)$," *Acta. Math. Hung.*, vol. 44, pp. 237–241, 1984.
[18] R. Lidl and H. Niederreiter, *Finite Fields*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1997.
[19] R. Lidl and G. L. Mullen, "When does a polynomial over a finite field permute the elements of the field ?," *Amer. Math. Month.*, vol. 95, pp. 243–246, Mar. 1988.
[20] R. Lidl and G. L. Mullen, "When does a polynomial over a finite field permute the elements of the field ? (II)," *Amer. Math. Month.*, vol. 100, pp. 71–74, Jan. 1993.

Soft-Decision Decoding of Reed–Muller Codes: Recursive Lists

Ilya Dumer, *Senior Member, IEEE,* and Kirill Shabunov, *Member, IEEE*

***Abstract*—Recursive list decoding is considered for Reed–Muller (RM) codes. The algorithm repeatedly relegates itself to the shorter RM codes by recalculating the posterior probabilities of their symbols. Intermediate decodings are only performed when these recalculations reach the trivial RM codes. In turn, the updated lists of most plausible codewords are used in subsequent decodings. The algorithm is further improved by using permutation techniques on code positions and by eliminating the most error-prone information bits. Simulation results show that for all RM codes of length 256 and many subcodes of length 512, these algorithms approach maximum-likelihood (ML) performance within a margin of 0.1 dB. As a result, we present tight experimental bounds on ML performance for these codes.**

***Index Terms*—Maximum-likelihood (ML) performance, Plotkin construction, posterior probabilities, recursive lists, Reed–Muller (RM) codes.**

I. INTRODUCTION

The main goal of this correspondence is to design feasible error-correcting algorithms that approach maximum-likelihood (ML) decoding on the moderate lengths ranging from 100 to 1000 bits. The problem is practically important due to the void left on these lengths by the best algorithms known to date. In particular, exact ML decoding has huge decoding complexity even on blocks of 100 bits. On the other hand,

Manuscript received June 4, 2004; revised November 15, 2005. This work was supported by the National Science Foundation under Grant CCR-0097125. The material in this correspondence was presented in part at the 38th Annual Allerton Conference on Communication, Control, and Computing, Monticello, IL, October 2000.

I. Dumer is with the College of Engineering, University of California, Riverside, CA 92521 USA (e-mail: dumer@ee.ucr.edu).

K. Shabunov is with the XVD Corporation, San Jose, CA 95134 USA (e-mail: kshabunov@xvdcorp.com).

Communicated by A. E. Ashikhmin, Associate Editor for Coding Theory.
Digital Object Identifier 10.1109/TIT.2005.864443

currently known iterative (message-passing) algorithms have been efficient only on blocks of thousands of bits.

To achieve near-ML performance with moderate complexity, we wish to use *recursive* techniques that repeatedly split an original code into the shorter ones. For this reason, we consider Reed–Muller (RM) codes, which represent the most notable example of recursive constructions known to date. These codes—denoted below as $\{ {m \atop r} \}$—have length $n = 2^m$ and Hamming distance $d = 2^{m-r}$. They also admit a simple recursive structure based on the *Plotkin construction* $(\boldsymbol{u}, \boldsymbol{u}+\boldsymbol{v})$, which splits the original RM code into the two shorter codes of length 2^{m-1}. This structure was efficiently used in recursive decoding algorithms of [2]–[4], which derive the corrupted symbols of the shorter codes $\boldsymbol{u}$ and $\boldsymbol{v}$ from the received symbols. These recalculations are then repeated until the process reaches the repetition codes or full spaces, whereupon new information symbols can be retrieved by any powerful algorithm—say, ML decoding. As a result, recursive algorithms achieve bounded distance decoding with a low complexity order of $n \min\{r, m-r\}$, which improves upon the complexity of majority decoding [1].

We also mention two list decoding algorithms of [5] and [6], which substantially reduce the error rates at the expense of a higher complexity. In both algorithms, RM codes are represented as the generalized concatenated codes, which are repeatedly decomposed into the shorter blocks similarly to the Plotkin construction. In all intermediate steps, the algorithm of [5] tries to estimate the Euclidean distance to the received vector and then retrieves the codewords with the smallest estimates. To do so, the algorithm chooses some number L of codewords from both constituent codes $\boldsymbol{u}$ and $\boldsymbol{v}$. Then the product list is constructed for the original code. These lists are recursively re-evaluated and updated in *multiple runs*. The second technique [6] is based on a novel sequential scheme that uses both the main stack and the complementary one. The idea here is to lower-bound the minimum distance between the received vector and the best code candidates that will be obtained in the *future steps*. This "look-ahead" approach gives low error rates and reduces the decoding complexity of [5].

Recently, new recursive algorithms were considered in [8] and [9]. In particular, for long RM codes of fixed code rate R, recursive decoding of [8] corrects most error patterns of weight $(d \ln d)/2$ instead of the former threshold of $d/2$. This is done without any increase in decoding complexity. However, the new decoding threshold is still inferior to that of a much more powerful ML decoding.

In the sequel, we advance the algorithm of [8], also applying list decoding techniques. This approach mostly follows [9] and differs from the prior results in a few important aspects. First, we use exact posterior probabilities in our recursive recalculations instead of the distance approximations employed before. This allows us to design a tree-like recursive algorithm that can better sort out all plausible candidates in intermediate steps and avoid multiple decoding runs. Second, we shall see that the output error rate significantly varies for the different information symbols derived in the recursive process. Therefore, we also consider subcodes of RM codes obtained by removing the least protected information bits. Finally, decoding will be improved by applying a few permutations on code positions. As a result, we closely approach the performance of ML decoding on the lengths 256 and 512, which was beyond the reach of the former techniques.

The material is organized as follows. In Section II, we briefly summarize some recursive properties of RM codes and their decoding procedures. In Section III, we describe our list decoding algorithm $\Psi_r^m(L)$. Finally, in Section IV, we discuss the improvements obtained by eliminating the least protected information bits and using permutation techniques.

II. Recursive Encoding and Decoding For RM Codes

A. Encoding

The following description is detailed in [10]. Let any codeword $\boldsymbol{c}$ of RM code $\{ {m \atop r} \}$ be represented in the form $\boldsymbol{u}, \boldsymbol{u}+\boldsymbol{v}$ where $\boldsymbol{u} \in \{ {m-1 \atop r} \}$ and $\boldsymbol{v} \in \{ {m-1 \atop r-1} \}$. We say that $\boldsymbol{c}$ is split onto two "paths" $\boldsymbol{u}$ and $\boldsymbol{v}$. By splitting both paths, we obtain four paths that lead to RM codes of length 2^{m-2}, and so on. In each step i of our splitting, we assign the path value $\xi_i = 0$ to a new $\boldsymbol{v}$-component and $\xi_i = 1$ to a new $\boldsymbol{u}$-component. All paths end at the repetition codes $\{ {g \atop 0} \}$ or full spaces $\{ {h \atop h} \}$, where

$$g = 1, \ldots, m-r, \quad h = 1, \ldots, r.$$

Thus, we can consider a specific binary path

$$\xi \stackrel{\text{def}}{=} (\xi_1, \ldots, \xi_{m-g})$$

that leads from the origin $\{ {m \atop r} \}$ to some left-end code $\{ {g \atop 0} \}$. For any right-end node $\{ {h \atop h} \}$, the same process gives a subpath ξ of length $m-h$

$$\xi \stackrel{\text{def}}{=} (\xi_1, \ldots, \xi_{m-h}).$$

A similar decomposition can be performed on the block $\boldsymbol{a}_r^m$ of k information bits that encode the original vector $\boldsymbol{c}$. In this way, any left-end path ξ gives only one information bit associated with its end node $\{ {g \atop 0} \}$. Any right-end path gives 2^h information bits associated with the end code $\{ {h \atop h} \}$. We can also add an arbitrary binary suffix of length h to the right-end paths, and obtain a one-to-one mapping between the extended paths ξ and k information bits $a(\xi)$.

B. Basic Decoding With Posterior Probabilities

Let any binary symbol a be mapped onto $(-1)^a$. Then any codeword of RM code belongs to $\{1, -1\}^n$ and has the form $\boldsymbol{c} = (\boldsymbol{u}, \boldsymbol{uv})$. This codeword is transmitted over a memoryless channel $\mathcal{Z}_g$. The received block $\boldsymbol{x}$ consists of the two halves $\boldsymbol{x}'$ and $\boldsymbol{x}''$, which are the corrupted images of vectors $\boldsymbol{u}$ and $\boldsymbol{uv}$. The decoder first takes the symbols x_i' and x_i'' for any position $i = 1, \ldots, n/2$, and finds the posterior probabilities of transmitted symbols u_i and $u_i v_i$

$$q_i' \stackrel{\text{def}}{=} \Pr\{u_i = 1 | x_i'\}, \quad q_i'' \stackrel{\text{def}}{=} \Pr\{u_i v_i = 1 | x_i''\}.$$

To simplify our notation, in the following we use the associated quantities

$$y_i' \stackrel{\text{def}}{=} 2q_i' - 1, \quad y_i'' = 2q_i'' - 1. \tag{1}$$

Note that y_i' is the *difference* between the two posterior probabilities q_i' and $1 - q_i'$ of 1 and -1 in position i of the left half. Similarly, y_i'' is obtained on the right half. The following basic recursive algorithm is described in [8] and [10, Sec. IV] in more detail.

Step 1. Let $q_i^v = \Pr\{v_i = 1 | x_i', x_i''\}$ be the posterior probability of any symbol v_i of the codeword $\boldsymbol{v}$. We find the corresponding quantity $y_i^v = 2q_i^v - 1$, which is (see [10, eq. (18)])

$$y_i^v = y_i' y_i''. \tag{2}$$

Symbols y_i^v form the vector $\boldsymbol{y}^v$ of length $n/2$. Then we use some soft-decision decoder $\Psi_v(\boldsymbol{y}^v)$ that gives a vector $\hat{\boldsymbol{v}} \in \{ {m-1 \atop r-1} \}$ and its information block $\hat{\boldsymbol{a}}^v$.

Step 2. Now we assume that Step 1 gives *correct vector* $\hat{\boldsymbol{v}} = \boldsymbol{v}$. Let $q_i^u = \Pr\{u_i = 1 | x_i', x_i''\}$ be the posterior probability of a symbol u_i. Then, the corresponding quantity $y_i^u = 2q_i^u - 1$ is (see [10, eq. (19)])

$$y_i^u = (y_i' + \hat{y}_i)/(1 + y_i'\hat{y}_i) \tag{3}$$

where $\hat{y}_i = y_i'' \hat{v}_i$. The symbols y_i^u form the vector $\boldsymbol{y}^u$ of length $n/2$. We use some (soft-decision) decoding algorithm $\Psi_u(\boldsymbol{y}^u)$ to obtain a vector $\hat{\boldsymbol{u}} \in \{ {m-1 \atop r} \}$ and its information block $\hat{\boldsymbol{a}}^u$. □

In a more general scheme Ψ_r^m, vectors $\boldsymbol{y}^v$ and $\boldsymbol{y}^u$ are not decoded but used as our new inputs $\boldsymbol{y}$. These inputs are recalculated multiple times according to (2) and (3). Finally, we reach the end nodes $\{ {g \atop 0} \}$ and $\{ {h \atop h} \}$. Here we perform ML decoding as follows.

At any node $\{ {g \atop h} \}$, our input is a newly recalculated vector $\boldsymbol{y}$ of length 2^g with the given differences y_i between posterior probabilities of two symbols $c_i = \pm 1$. Rewriting definition (1), we assign the posterior probability

$$\Pr(c_i | y_i) = (1 + c_i y_i)/2$$

to a symbol $c_i = \pm 1$. In this way, we can find the posterior probability

$$P(\boldsymbol{c} \mid \boldsymbol{y}) = \prod_{i=1}^{2^g} (1 + c_i y_i)/2 \tag{4}$$

of any codeword $\boldsymbol{c} \in \{ {g \atop h} \}$, and choose the most probable codeword $\hat{\boldsymbol{c}}$, where

$$\forall \boldsymbol{c} \in \left\{ \begin{matrix} g \\ h \end{matrix} \right\} : P(\hat{\boldsymbol{c}} | \boldsymbol{y}) \geq P(\boldsymbol{c} \mid \boldsymbol{y}). \tag{5}$$

The decoded codeword $\hat{\boldsymbol{c}} \in \{ {m \atop r} \}$ and the corresponding information block $\hat{\boldsymbol{a}}$ are now obtained as follows (here operations (2) and (3) are performed on vectors componentwise).

Algorithm Ψ_r^m for an input vector $\boldsymbol{y}$.

1. If $0 < r < m$, execute the following.
   1.1. Calculate vector $\boldsymbol{y}^v = \boldsymbol{y}'\boldsymbol{y}''$.
   Decode $\boldsymbol{y}^v$ into vector $\hat{\boldsymbol{v}} = \Psi_{r-1}^{m-1}(\boldsymbol{y}^v)$.
   Pass $\hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{a}}^v$ to Step 1.2
   1.2. Calculate vector $\boldsymbol{y}^u = (\boldsymbol{y}' + \hat{\boldsymbol{y}})/(1 + \boldsymbol{y}'\hat{\boldsymbol{y}})$.
   Decode $\boldsymbol{y}^u$ into vector $\hat{\boldsymbol{u}} = \Psi_r^{m-1}(\boldsymbol{y}^u)$.
   Output decoded components:
   $\hat{\boldsymbol{a}} := (\hat{\boldsymbol{a}}^v \mid \hat{\boldsymbol{a}}^u)$; $\hat{\boldsymbol{c}} := (\hat{\boldsymbol{u}} \mid \hat{\boldsymbol{u}}\hat{\boldsymbol{v}})$.
2. If $r = 0$, use ML decoding (5) for $\{ {r \atop 0} \}$.
3. If $r = m$, use ML decoding (5) for $\{ {r \atop r} \}$.

Note that this algorithm Ψ_r^m differs from the simplified algorithm Φ_r^m of [10] in three aspects. First, we use exact recalculations (3) instead of the former simplification

$$\boldsymbol{y}^u = (\boldsymbol{y}' + \hat{\boldsymbol{y}})/2. \tag{6}$$

Second, we use ML decoding instead of the minimum distance decoding that chooses $\hat{\boldsymbol{c}}$ with the maximum inner product

$$\forall \boldsymbol{c} : (\hat{\boldsymbol{c}}, \boldsymbol{y}) \geq (\boldsymbol{c}, \boldsymbol{y})$$

Third, we employ a different rule and stop at the repetition codes $\{ {r \atop 0} \}$ instead of the biorthogonal codes used in [10]. This last change will make it easier to use the list decoding described in the following section.

Finally, note that recalculations (2) require $n/2$ operations, while recalculations (3) can be done in $5n/2$ operations. Therefore, our decoding complexity satisfies recursion

$$|\Psi_r^m| \leq \left|\Psi_{r-1}^{m-1}\right| + \left|\Psi_r^{m-1}\right| + 3n.$$

Similarly to [10], this recursion gives decoding complexity

$$|\Psi_r^m| \leq 6n \min(r, m - r) + n.$$

Thus, complexity $|\Psi_r^m|$ has maximum order of $3n \log n$, which is twice the complexity $|\Phi_r^m|$ of the algorithm Φ_r^m of [10].

III. List Decoding

To enhance algorithm Ψ_r^m, we shall use some lists of $L = 2^p$ or fewer codewords obtained on any path ξ. This algorithm—called $\Psi_r^m(L)$—increases the number of operations at most L times and has the overall complexity order of $Ln \log n$. Given any integer parameter A, we say that the list has size A^*, if decoding outputs either all available records or A records, whichever is less. This algorithm performs as follows.

At any step $s = 1, \ldots, k$ of the algorithm $\Psi_r^m(L)$, our input consists of L^* records

$$A = (\bar{\boldsymbol{a}}, \rho(\bar{\boldsymbol{a}}), \boldsymbol{y}(\bar{\boldsymbol{a}})).$$

Each record is formed by some information block $\bar{\boldsymbol{a}}$, its cost function $\rho(\bar{\boldsymbol{a}})$, and the corresponding input $\boldsymbol{y}(\bar{\boldsymbol{a}})$, which is updated in the decoding process. These three entries are defined in the following.

Decoding starts at the root node $\{ {m \atop r} \}$. Here we set $s = 0$ and take one record

$$\bar{\boldsymbol{a}} = \emptyset, \quad \rho(\bar{\boldsymbol{a}}) = 1, \quad \boldsymbol{y}(\bar{\boldsymbol{a}}) = \boldsymbol{y} \tag{7}$$

where $\boldsymbol{y}$ is the input vector. Decoding takes the first path (denoted $\xi = 1$) to the leftmost code $\{ {m-r \atop 0} \}$ and recalculates vector $\boldsymbol{y}(\bar{\boldsymbol{a}})$ similarly to the algorithm Ψ_r^m. However, now we take *both* values $a_1 = 0, 1$ of the first information symbol and consider both codewords 1^d and -1^d of length $d = 2^{m-r}$ in the repetition code $\boldsymbol{c}(a_1)$. The posterior probabilities (4) of these two vectors will also define the cost function of the new information block $\bar{\boldsymbol{a}} = a_1$

$$\rho(\bar{\boldsymbol{a}}) = \prod_{i=1}^{2^{m-r}} \frac{1 + \boldsymbol{c}_i(a_1) y_i(\bar{\boldsymbol{a}})}{2}.$$

In our list decoding, we represent the two outcomes $\bar{\boldsymbol{a}}$ as the initial edges mapped with their cost functions $P(\bar{\boldsymbol{a}})$. Then we proceed to the next code $\{ {m-r-1 \atop 0} \}$, which corresponds to the subsequent path denoted $\xi = 2$. Given two different decoding results $\boldsymbol{v} = \boldsymbol{c}(a_1)$, our recursion (2), (3) gives two different vectors $\boldsymbol{y}(\bar{\boldsymbol{a}})$ arriving at this node. Therefore, decoding is performed two times and gives the full tree of depth 2. More generally, at any step s, decoding is executed as follows.

Suppose that the first $s - 1$ paths are already processed. This gives L^* information blocks

$$\bar{\boldsymbol{a}} = (a_1, \ldots, a_{s-1})$$

of length $s - 1$ and the corresponding records A. Each vector $\boldsymbol{y}(\bar{\boldsymbol{a}})$ is then recalculated on the new path $\xi = s$ using formulas (2) and (3) in the same way it was done in Ψ_r^m. Let this path end on some left-end code $\{ {g \atop 0} \}$. Decoding of the new information symbol $a_s = 0, 1$ yields $2L^*$ extended blocks

$$\bar{\boldsymbol{a}} := \bar{\boldsymbol{a}}, a_s$$

of depth s, marked by their cost functions

$$\rho(\bar{\boldsymbol{a}}) := \rho(\bar{\boldsymbol{a}}) \cdot \prod_{i=1}^{2^g} \frac{1 + c_i(a_s) y_i(\bar{\boldsymbol{a}})}{2}. \tag{8}$$

Step s is completed by choosing L^* blocks with the highest cost functions $\rho(\bar{\boldsymbol{a}})$.

The decoding on the right-end nodes is similar. The only difference is that the full spaces $\{ {h \atop h} \}$ include 2^h codewords defined by information blocks a_s of length $|a_s| = h$. In this case, we can choose the two most probable vectors $\boldsymbol{c}(a_s)$ (in essence, making bit-by-bit decisions) and set $g = h$ in our cost calculations (8). Another—more refined version of the algorithm—chooses four different vectors of the code $\{ {h \atop h} \}$ whenever $h \geq 2$. The best record is chosen at the last node $\{ {r \atop r} \}$. More generally, the algorithm is executed as follows.

Algorithm $\Psi_r^m(L)$. Input: L^* records $A = (\bar{\boldsymbol{a}}, \rho(\bar{\boldsymbol{a}}), \boldsymbol{y}(\bar{\boldsymbol{a}}))$, counter $s = 0$.
1. If $0 < r < m$, for all vectors $\boldsymbol{y}(\bar{\boldsymbol{a}})$:
   1.1. Set $y(\bar{\boldsymbol{a}}) := \boldsymbol{y}'(\bar{\boldsymbol{a}})\boldsymbol{y}''(\bar{\boldsymbol{a}})$.
   Perform decoding $\Psi_{r-1}^{m-1}(\boldsymbol{y}(\bar{\boldsymbol{a}}))$.
   Pass L^* new records A to Step 1.2
   1.2. Set $\boldsymbol{y}(\bar{\boldsymbol{a}}) := \dfrac{\boldsymbol{y}'(\bar{\boldsymbol{a}}) + \hat{\boldsymbol{y}}(\bar{\boldsymbol{a}})}{1 + \boldsymbol{y}'(\bar{\boldsymbol{a}})\hat{\boldsymbol{y}}(\bar{\boldsymbol{a}})}$.
   Perform decoding $\Psi_r^{m-1}(\boldsymbol{y}(\bar{\boldsymbol{a}}))$.
   Output L^* new records A.
2. If $r = 0$, take both values $a_s = 0, 1$.
Calculate costs (8) for each $(\bar{\boldsymbol{a}}, a_s)$.
Choose L^* best blocks $\bar{\boldsymbol{a}} := (\bar{\boldsymbol{a}}, a_s)$.
Set $s := s + 1$ and return L^* records A.
3. If $r = m$, choose 4^* best blocks a_s.
Calculate costs (8) for each $(\bar{\boldsymbol{a}}, a_s)$.
Choose L^* best blocks $\bar{\boldsymbol{a}} := (\bar{\boldsymbol{a}}, a_s)$.
Set $s := s + |a_s|$ and return L^* records A.

Discussion: Consider the above algorithm on the additive white Gaussian noise (AWGN) channel $\mathcal{N}(0, \sigma^2)$. Using the results of [10], it can be proven that on this channel, the $\boldsymbol{v}$-component is decoded on the channel with the new noise power

$$\sigma_v^2 \geq \max\{2\sigma^2, \sigma^4\}.$$

The first approximation is tight for very small σ^2 (though the channel is no longer Gaussian), while the second one performs well on the "bad" channels with $\sigma^2 \gg 1$. Thus, the noise power always increases in the $\boldsymbol{v}$-direction; the more so, the worse the original channel is. By contrast, the $\boldsymbol{u}$-channel can be approximated by the smaller power $\sigma^2/2$. These observations also show that the first information symbol—which is obtained on the binary path 0^r—is protected the least, and then the decoding gradually improves on the subsequent paths.

Now we see that the algorithm $\Psi_r^m(L)$ with the list of size $L = 2^p$ delays our decision on any information symbol by p steps, making this decision better protected. In the particular case of a bad channel, it can be verified that the first symbol a_1 is now decoded when the noise power is reduced 2^p times. For this reason, this list decoding substantially reduces the output word-error rates (WER) even for small size L.

For $L = 2^{m-r+1}$, the algorithm $\Psi_r^m(L)$ processes all the codewords of the first biorthogonal code $\{ {m-r+1 \atop 1} \}$ and is similar to the algorithm Φ_r^m of [10]. On the other hand, algorithm $\Psi_r^m(L)$ updates all L cost functions, while Φ_r^m chooses one codeword on each end node. Therefore, $\Psi_r^m(L)$ can be considered as a generalization of Φ_r^m that continuously updates decoding lists in all intermediate steps. The result is a more powerful decoding that comes along with a higher complexity.

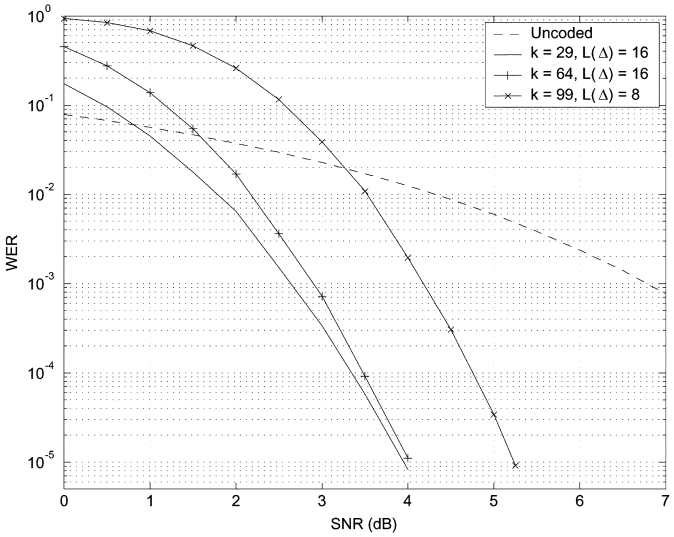

Fig. 1. Tight lower bounds on WER of ML decoding for three RM codes of length 128. The legend gives the list size $L(\Delta)$ for which the algorithm $\Psi_r^m(L)$ performs withing $\Delta = 0.25$ dB from these bounds.

TABLE I
RM CODES OF LENGTH 128: THE LIST SIZE $L(\Delta)$, DECODING COMPLEXITY, AND THE CORRESPONDING SNR AT WHICH ALGORITHM $\Psi_r^m(L)$ PERFORMS WITHIN $\Delta = 0.25$ dB FROM ML DECODING AT WER 10^{-4}

RM Code	$\{ {7 \atop 2} \}$	$\{ {7 \atop 3} \}$	$\{ {7 \atop 4} \}$
List size $L(\Delta)$	16	16	8
Complexity $\lvert\Psi(L)\rvert$	21676	33618	18226
SNR (dB) at 10^{-4}	3.47	3.71	4.85

Simulation Results: In the following, we present our simulation results for the AWGN channels. Here we also counted all the instances, when for a given output the decoded vector was more probable than the transmitted one. Obviously, these specific events also represent the errors of ML decoding. Thus, the fraction of these events gives a lower bound on the ML decoding error probability. This lower bound is also depicted in the subsequent figures for all the codes tested.

Our simulation results show that for all RM codes of lengths 128 and 256, the algorithm $\Psi_r^m(L)$ rapidly approaches ML performance as the list size L grows. For RM codes of length 128 and distance $d > 4$, we summarize these results in Fig. 1. For each RM code, we present *tight* lower bounds for the error probability of ML decoding. To measure the efficiency of the algorithm $\Psi_r^m(L)$, we also exhibit the actual list size $L(\Delta)$ at which $\Psi_r^m(L)$ approaches the optimal ML decoding within a small margin of

$$\Delta = 0.25 \text{ dB}.$$

This performance loss Δ is measured at the output WER $P = 10^{-4}$; however, we found little to no difference for all other WER tested in our simulation. In Table I, we complement these sizes $L(\Delta)$ with the two other relevant parameters:

— the signal-to-noise ratios (SNR per information bit) at which algorithm $\Psi_r^m(L)$ gives the WER $P = 10^{-4}$;

— the complexity estimates $|\Psi_r^m(L)|$ counted as the number of floating-point operations.

For RM codes of length 256, we skip most decoding results as these will be improved in the next section by the permutation techniques. In our single example in Fig. 2, we present the results for the $(n = 256, k = 93)$ code $\{{8 \atop 3}\}$. This code gives the *lowest* rate of convergence to the ML decoding among all RM codes of length 256. In other words, all other codes require the smaller lists to achieve the same performance loss Δ. This example and other simulation results show that the algorithm $\Psi_r^m(L)$ performs within 0.5 dB from ML decoding on the lengths 128 and 256 using lists of small or moderate size.

IV. Further Improvements

A. Subcodes of RM Codes

More detailed results also show that many codes of length $n \geq 256$ require lists of large size $L \geq 1024$ to approach ML decoding within the small margin of 0.25 dB. Therefore, for $n \geq 256$, we also employ a different approach. Namely, the decoding performance can be improved by eliminating those paths, where recursive decoding fails more often. Here we use the results of [10], which show that the left-most paths are the least protected.

Recall that each left-end path ξ corresponds to one information symbol. Therefore, decoding on these paths can be eliminated by *setting the corresponding information bits as zeros*. In this way, we employ the *subcodes* of the original code $\{{m \atop r}\}$. Note that our decoding algorithm $\Psi_r^m(L)$ runs virtually unchanged on subcodes. Indeed, the single difference arises when some information block a_s takes only one value 0 on the corresponding left node (or less than 2^h values on the right node). Therefore, on each step s, we can proceed as before, by taking only the actual blocks a_s left at this node after expurgation.

In the algorithm $\Psi_r^m(L)$, this expurgation starts with the least protected information path 0^r that ends at the node $\{{m-r \atop 0}\}$. It can be shown that for long RM codes of fixed order r, eliminating even the single weakest path 0^r increases the admissible noise power $2^{1/2^r}$ times. Thus, the lowest orders $r = 2, 3$ yield the biggest gain $(10\log_{10} 2)/2^r$ dB, which equals 0.75 and 0.375 dB, respectively.

To proceed further, we eliminate the next weakest path $0^{r-1}10$. However, the theoretical analysis becomes more complicated on the subsequent bits and it is unclear which bits should be eliminated first. For this reason, we optimized this pruning process in our simulation by making a few *ad hoc* trials and eliminating subsequent bits in different order.

The corresponding simulation results are presented in Fig. 3 for the $(256, 93)$-code $\{{8 \atop 3}\}$ and its $(256, 78)$-subcode. We see that pruning substantially improves code performance. It is also interesting to compare Figs. 2 and 3. We see that the subcode approaches the optimal ML performance much faster than the original code does. In particular, the same margin of $\Delta = 0.25$ dB can be reached with only $L = 16$ codewords instead of $L = 1024$ codewords needed on the code. In all other examples, the subcodes also demonstrated a much faster convergence, which leads to a lesser complexity.

In Fig. 4, we present similar results for the $(512, 101)$-subcode of the $(512, 130)$-code $\{{9 \atop 3}\}$. Here in Table II, we also give a few list sizes L, the corresponding SNRs needed to reach the output WER $P = 10^{-4}$, and the complexity estimates $|\Psi_r^m(L)|$ counted by the number of floating-point operations. Similar results were also obtained for the subcodes of other RM codes of length 512.

These simulation results show that combining both techniques—eliminating the least protected bits and using small lists of code-words—gives a gain of 3 to 4 dB on the lengths $n \leq 512$ over the original non-list decoding algorithm Ψ_r^m. For subcodes, we also approach ML decoding with the lists reduced up to 64 times relative to the original RM codes.

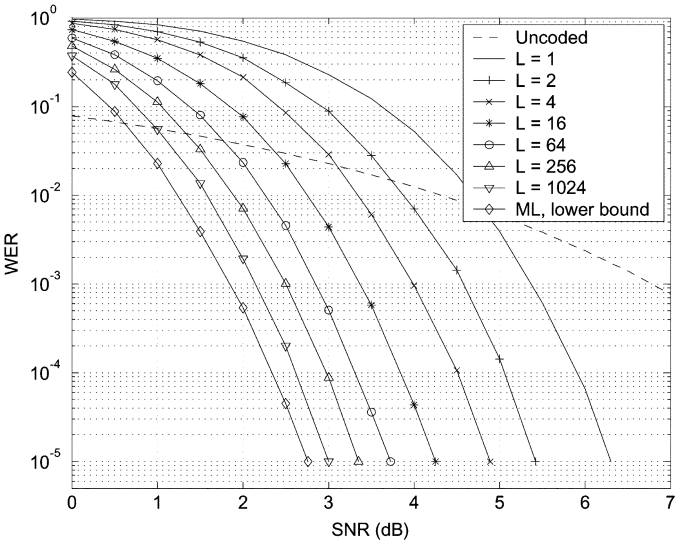

Fig. 2. $(256, 93)$ RM code $\{{8 \atop 3}\}$. WER for the algorithm $\Psi_r^m(L)$ with lists of size L.

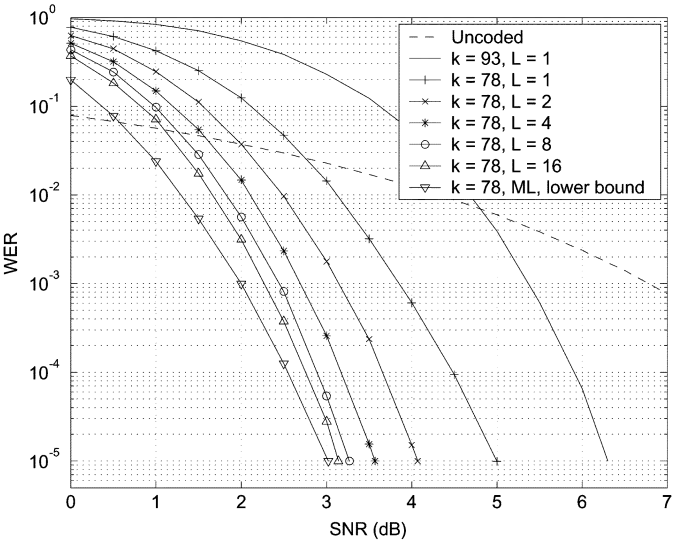

Fig. 3. $(256, 78)$-subcode of the $(256, 93)$ RM code $\{{8 \atop 3}\}$. WER for the algorithm $\Psi_r^m(L)$ with lists of size L.

B. New Permutation Techniques

The second improvement to the algorithm $\Psi_r^m(L)$ utilizes the rich *symmetry group* $GA(m)$ of RM codes [7] that includes $2^{O(m^2)}$ permutations of n positions $i = (i_1, \ldots, i_m)$. To employ fewer permutations, we first permute the m indices $(1, 2, \ldots, m)$ of all n positions $i = (i_1, \ldots, i_m)$. Thus, we first take a permutation

$$(1, 2, \ldots, m) \stackrel{\pi}{\mapsto} (\pi(1), \ldots, \pi(m))$$

of m *indices* and consider the corresponding $m!$ permutations $\pi(i)$ of positions i

$$\pi(i) : (i_1, \ldots, i_m) \to \left(i_{\pi(1)}, \ldots, i_{\pi(m)}\right). \tag{9}$$

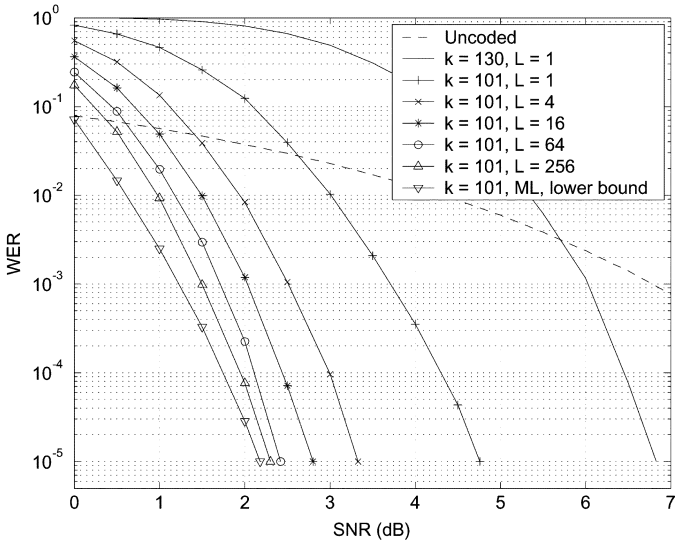

Fig. 4. $(512, 101)$-subcode of the $(512, 130)$ RM code $\{{}^{9}_{3}\}$. WER for the algorithm $\Psi_r^m(L)$ with lists of size L.

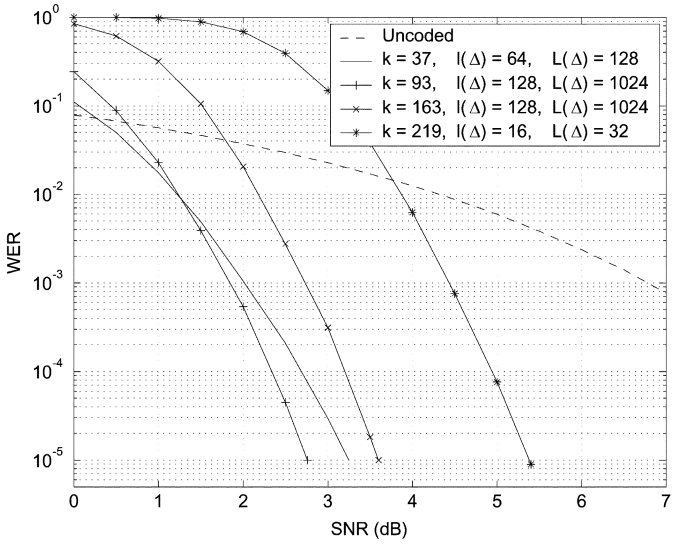

Fig. 5. Tight lower bounds on WER of ML decoding for four RM codes of length 256. The legend gives the list sizes $l(\Delta)$ and $L(\Delta)$ for which the algorithms $\Upsilon_r^m(l)$ and $\Psi_r^m(L)$ perform within $\Delta = 0.25$ dB from these bounds.

TABLE II
$(512, 101)$-SUBCODE OF THE $(512, 130)$ RM CODE $\{{}^{9}_{3}\}$. LIST SIZES L, THE CORRESPONDING SNRS, AND COMPLEXITY ESTIMATES $|\Psi_r^m(L)|$ NEEDED AT WER 10^{-4}

List size L	1	4	16	64
Complexity	7649	25059	92555	378022
SNR at 10^{-4}	4.31	3	2.5	2.1

Remark: Note that the m indices represent the different *axes* in E_2^m. Thus, any permutation of indices is the permutation of axes of E_2^m. For example, the permutation $(2, 1, 3, 4, \ldots, m)$ of m indices leaves unchanged the first and the fourth quarters of all positions $1, \ldots, n$, but changes the order of the second and the third quarters.

Given a permutation π, consider the subset of r original indices (axes) $\pi^{-1}\{1, \ldots, r\}$ that were transformed into the first r axes $1, \ldots, r$ by the permutation π. We say that two permutations π and η are equivalent if these images form the identical (unordered) subsets

$$\pi^{-1}\{1, \ldots, r\} = \eta^{-1}\{1, \ldots, r\}.$$

Now consider any subset T of permutations (9) that includes exactly one permutation from each equivalent class. Thus, T includes $\binom{m}{r}$ permutations, each of which specifies a subset of the first r indices. Recall that these r indices correspond to the axes that are processed first on the subpath 0^r (for example, we can start with the axis i_2 instead of i_1, in which case we first fold the adjacent quarters instead of the halves of the original block). Thus, this subset T specifies all possible ways of choosing r unordered axes that will be processed first by the algorithm Ψ_r^m.

Given some positive integer l (which is smaller than the former parameter L), we then incorporate these permutations $\pi(i)$ into the list decoding $\Psi_r^m(l)$. Namely, we form all permutations $\boldsymbol{y}_{\pi(i)}$ of the received vector $\boldsymbol{y}$ and apply algorithm $\Psi_r^m(l)$ to each vector $\boldsymbol{y}_{\pi(i)}$. However, at each step of the algorithm, we also *combine different* lists and leave only l best candidates in the combined list, each counted once.

Note that this technique makes only marginal changes to our conventional list decoding $\Psi_r^m(l)$. Indeed, the single vector $\boldsymbol{y}$ in our original setting (7) is replaced by $\binom{m}{r}$ permutations $\boldsymbol{y}_{\pi(i)}$. Thus, we use parameter $\binom{m}{r}$ in our initial setting but keep parameter l for all decoding steps. If $l < \binom{m}{r}$, then the number of records drops to l almost immediately, after the first decoding is performed on the path 0^r.

Also, information bits are now decoded in different orders depending on a specific permutation $\pi(i)$. Note that we may (and often do) get the same entries repeated many times. Therefore, in Steps 2 and 3 we must eliminate identical entries. This is done in all steps by applying inverse permutations and comparing the corresponding blocks $\boldsymbol{a}$. This permutation-based algorithm is called $\Upsilon_r^m(l)$ below and has complexity similar to $|\Psi_r^m(l)|$ for all the codes tested.

The motivation for this algorithm is as follows. The specific order of our axes also defines the order in which the decoding algorithm folds the original block into the subblocks of lengths $n/2$, then $n/4$, and so on. Now note that this folding procedure will likely accumulate the errors whenever erroneous positions substantially disagree on the two halves (correspondingly, quarters, and so on). This can also happen if the errors are unevenly spread over the two halves of the original block. By using many permutations, we make it more likely that the error positions are spread more evenly even if they get accumulated in the original setting $\pi(i) = i$ or any other specific setting. In this way, permutation techniques serve the same functions as interleaving does on the bursty channels.

Simulation results for the moderate lengths 256 and 512 show that the algorithm $\Upsilon_r^m(l)$ approaches the optimal ML performance even when the combined list of l most probable candidates *is reduced two to eight times* relative to the original algorithm $\Psi_r^m(L)$. For RM codes of length 256, we summarize these results in Fig. 5. For each RM code, we first present the lower bounds for the ML decoding error probability. Similarly to Fig. 1, we then find the minimum size $l(\Delta)$ that makes the algorithm $\Upsilon_r^m(l)$ perform only within $\Delta = 0.25$ dB away from ML decoding. These sizes and complexity estimates $|\Upsilon_r^m(l)|$ are also given in Table III. Note that both algorithms give smaller lists once this performance loss Δ is slightly increased. In particular, the results in Table IV show that the lists are reduced two times for $\Delta = 0.5$ dB.

In summary, the permutation algorithm $\Upsilon_r^m(l)$ *performs within 0.5 dB from ML decoding on the length* 256, by processing $l \leq 64$ vectors for all RM codes. To date, both techniques—permutation decoding $\Upsilon_r^m(l)$ of complete RM codes and list decoding $\Psi_r^m(L)$ of their subcodes—yield the best tradeoffs between near-ML performance and its complexity known on the lengths $n \leq 256$.

TABLE III
RM CODES OF LENGTH 256: THE LIST SIZES, COMPLEXITIES, AND THE CORRESPONDING SNRS, AT WHICH THE PERMUTATION ALGORITHM $\Upsilon_r^m(l)$ PERFORMS WITHIN $\Delta = 0.25$ dB FROM ML DECODING AT WER 10^{-4}

RM Code	$\{^8_2\}$	$\{^8_3\}$	$\{^8_4\}$	$\{^8_5\}$
List size $l(\Delta)$	64	128	128	16
Complexity $\lvert\Upsilon_r^m(l)\rvert$	216752	655805	777909	94322
SNR at 10^{-4}	2.91	2.65	3.38	5.2

TABLE IV
RM CODES OF LENGTH 256: THE LIST SIZES, COMPLEXITIES, AND THE CORRESPONDING SNRS, AT WHICH THE PERMUTATION ALGORITHM $\Upsilon_r^m(l)$ PERFORMS WITHIN $\Delta = 0.5$ dB FROM ML DECODING AT WER 10^{-4}

RM Code	$\{^8_2\}$	$\{^8_3\}$	$\{^8_4\}$	$\{^8_5\}$
List size $l(\Delta)$	32	64	64	8
Complexity $\lvert\Upsilon_r^m(l)\rvert$	116471	333506	389368	37756
SNR at 10^{-4}	3.12	2.82	3.55	5.4

Note, however, that the algorithm $\Upsilon_r^m(l)$ gives almost no advantage for the subcodes considered in the previous subsection. Indeed, these subcodes are obtained by eliminating the leftmost (least protected) information bits. However, any new permutation $\pi(i)$ assigns the new information bits to these leftmost nodes. Thus, the new bits also become the least protected. Another unsatisfactory observation is that increasing the size of the permutation set T—say, to include all $m!$ permutations of all m indices—helps little in improving decoding performance. More generally, there are a number of important open problems related to these permutation techniques. We name a few:

— find the best permutation set T for the algorithm $\Upsilon_r^m(l)$;
— analyze the algorithm $\Upsilon_r^m(l)$ analytically;
— modify the algorithm $\Upsilon_r^m(l)$ for subcodes.

V. CONCLUDING REMARKS

In this correspondence, we considered recursive decoding algorithms for RM codes that can provide near-ML decoding with feasible complexity for RM codes or their subcodes on the moderate lengths $n \leq 512$.

Our study still leaves many open problems. First, we need to tightly estimate the error probabilities $p(\xi)$ on the different paths ξ. To optimize our pruning procedures for specific subcodes, it is important to find the order in which information bits should be removed from the original RM code. Finally, it is still an open problem to analytically estimate the performance of the algorithms $\Psi_r^m(L)$ and $\Upsilon_r^m(l)$.

ACKNOWLEDGMENT

The authors wish to thank an anonymous referee for helpful suggestions.

REFERENCES

[1] I. S. Reed, "A class of multiple error correcting codes and the decoding scheme," *IEEE Trans. Inf. Theory*, vol. IT-4, no. 4, pp. 38–49, Sep. 1954.
[2] S. N. Litsyn, "On decoding complexity of low-rate Reed-Muller codes" (in Russian), in *Proc. 9th All-Union Conf. Coding Theory and Information Transmission*, Odessa, Ukraine, U.S.S.R., 1988, pp. 202–204.
[3] F. Hemmati, "Closest coset decoding of $u|u+v|$ codes," *IEEE Sele. Areas Commun.*, vol. 7, no. 6, pp. 982–988, Aug. 1989.
[4] G. A. Kabatyanskii, "On decoding of Reed-Muller codes in semicontinuous channels," in *Proc. 2nd Int. Workshop "Algebraic and Combinatorial Coding Theory"*, Leningrad, Russia, U.S.S.R., 1990, pp. 87–91.
[5] R. Lucas, M. Bossert, and A. Dammann, "Improved soft-decision decoding of reed-muller codes as generalized multiple concatenated codes," in *Proc. ITG Conf. Source and Channel Coding*, Aahen, Germany, 1998, pp. 137–141.
[6] N. Stolte and U. Sorger, "Soft-decision stack decoding of binary Reed-Muller codes with "Look-Ahead'" technique," in *Proc. 7th Int. Workshop "Algebraic and Combinatorial Coding Theory"*, Bansko, Bulgaria, Jun. 18–24, 2000, pp. 293–298.
[7] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*. Amsterdam, The Netherlands: North-Holland, 1981.
[8] I. Dumer, "Recursive decoding of Reed-Muller codes," in *Proc. 37th Allerton Conf. Communications, Control, and Computing*, Monticello, IL, Sep. 1999, pp. 61–69.
[9] I. Dumer and K. Shabunov, "Recursive constructions and their maximum likelihood decoding," in *Proc. 38th Allerton Conf. on Communications, Control, and Computing*, Monticello, IL, Oct. 2000, pp. 71–80.
[10] I. Dumer, "Soft decision decoding of Reed–Muller codes: A simplified algorithm," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 954–963, Mar. 2006.

Some Restrictions on Weight Enumerators of Singly Even Self-Dual Codes

Masaaki Harada and Akihiro Munemasa

***Abstract*—In this correspondence, we give some restrictions on weight enumerators of singly even self-dual $[n, n/2, d]$ codes whose shadows have minimum weight $d/2$. As a consequence, we determine the weight enumerators for which there is an extremal singly even self-dual $[40, 20, 8]$ code and an optimal singly even self-dual $[50, 25, 10]$ code.**

***Index Terms*—Extremal code, minimum weight, self-dual code, shadow, weight enumerator.**

I. INTRODUCTION

Let C be a singly even self-dual code and let C_0 denote the subcode of codewords having weight $\equiv 0 \pmod 4$. Then C_0 is a subcode of codimension 1. The *shadow* S of C is defined to be $C_0^{\perp} \setminus C$. Shadows for self-dual codes were introduced by Conway and Sloane [1] in order to derive new upper bounds for the minimum weight of singly even self-dual codes, and to provide restrictions on the weight enumerators of singly even self-dual codes. Using shadows, the largest possible minimum weights of singly even self-dual codes of lengths up to 72 are determined in [1, Table I]. The work was extended to lengths up to 100 in [2, Table VI]. The possible weight enumerators of singly even self-dual codes with the largest possible minimum weights are

Manuscript received March 29, 2005; revised September 28, 2005.

M. Harada is with the Department of Mathematical Sciences, Yamagata University, Yamagata 990-8560, Japan.
A. Munemasa is with the Graduate School of Information Sciences, Tohoku University, Sendai 980-8579, Japan.

Communicated by A. E. Ashikhmin, Associate Editor for Coding Theory.
Digital Object Identifier 10.1109/TIT.2005.864416