Title
Soft-decision decoding of Reed-Muller codes: Recursive lists

Permalink
https://escholarship.org/uc/item/4326p802

Journal
IEEE Transactions on Information Theory, 52(3)

ISSN
0018-9448

Authors
Dumer, I
Shabunov, K

Publication Date
2006-03-01

Peer reviewed
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,
   
   $2^n N_{2} \leq \left\lfloor 2^n q^2 \cdot 2^n q^2 \cdot 2^n q^2 \right\rfloor$.

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
   
   $2^n N_{2} \leq \left\lfloor 2^n q^2 \cdot 2^n q^2 \cdot 2^n q^2 \right\rfloor$.

3) $n_{N, 2} \geq 5$
   
   If $n_{N, 2} \geq 5$, then
   
   $\left\lfloor \frac{n_{N, 2} - 2}{2} \right\rfloor > 1$.

   By Lemma 3.3, if $n_{N, 2} - 1 > n_{F, 2} \geq \left\lfloor \frac{n_{N, 2} - 2}{2} \right\rfloor$, 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, $2^n N_{2} \leq \left\lfloor 2^n q^2 \cdot 2^n q^2 \cdot 2^n q^2 \right\rfloor$. 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 $2^n N_{2} \leq \left\lfloor 2^n q^2 \cdot 2^n q^2 \cdot 2^n q^2 \right\rfloor$.

\section*{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.

\section*{REFERENCES}


\section*{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.

\section*{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,
II. RECURSIVE ENCODING AND DECODING FOR RM CODES

A. Encoding

The following description is detailed in [10]. Let any codeword \( \mathbf{e} \) of RM code \( \{ m \} \) be represented in the form \( \mathbf{u}, \mathbf{u} + \mathbf{v} \) where \( \mathbf{u} \in \{ m-1 \} \) and \( \mathbf{v} \in \{ m-1 \} \). We say that \( \mathbf{e} \) is split onto two “paths” \( \mathbf{u} \) and \( \mathbf{v} \). 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 \( \mathbf{v} \)-component and \( \xi_i = 1 \) to a new \( \mathbf{u} \)-component. All paths end at the repetition codes \( \{ n \} \) or full spaces \( \{ n \} \), where

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

Thus, we can consider a specific binary path

\[
\xi \in \{ \xi_1, \ldots, \xi_{m-r} \}
\]

that leads from the origin \( \{ m \} \) to some left-end code \( \{ n \} \). For any right-end node \( \{ n \} \), the same process gives a subpath \( \xi \) of length \( m-h \)

\[
\xi \in \{ \xi_1, \ldots, \xi_{m-h} \}.
\]

A similar decomposition can be performed on the block \( a^{m} \) of \( k \) information bits that encode the original vector \( \mathbf{e} \). In this way, any left-end path \( \xi \) gives only one information bit associated with its end node \( \{ n \} \).

Any right-end path gives \( a^{h} \) information bits associated with the end code \( \{ n \} \). 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 \( \xi \) and \( k \) information bits \( a(\xi) \).

B. Basic Decoding With Posterior Probabilities

Let any binary symbol \( b \) be mapped onto \( (-1)^{b} \). Then any codeword of RM code belongs to \( \{ 1, -1 \}^{m} \) and has the form \( \mathbf{e} = \{ \mathbf{u}, \mathbf{v} \} \). This codeword is transmitted over a memoryless channel \( \mathbf{Z} \). The received block \( \mathbf{y} \) consists of the two halves \( x^{L}_{i} \) and \( x^{R}_{i} \), which are the corrupted images of vectors \( \mathbf{u} \) and \( \mathbf{v} \). The decoder first takes the symbols \( x^{L}_{i} \) and \( x^{R}_{i} \) for any position \( i = 1, \ldots, m/2 \), and finds the posterior probabilities of transmitted symbols \( u_{i} \) and \( u_{i}\).

\[
q^{(u)}_{i} \overset{\text{def}}{=} \Pr \{ u_{i} = 1 | x^{L}_{i} \}, \quad q^{(u)}_{i} \overset{\text{def}}{=} \Pr \{ u_{i}, x^{R}_{i} = 1 \}.
\]

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

\[
y^{(u)}_{i} \overset{\text{def}}{=} 2q^{(u)}_{i} - 1, \quad y^{(u)}_{i} \overset{\text{def}}{=} 2q^{(u)}_{i} - 1.
\]

Note that \( y^{(u)}_{i} \) is the difference between the two posterior probabilities \( q^{(u)}_{i} \) and \( 1 - q^{(u)}_{i} \) of 1 and \( -1 \) in position \( i \) of the left half. Similarly, \( y^{(u)}_{i} \) 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 \( q^{(u)}_{i} = \Pr \{ v_{i} = 1 | x^{L}_{i}, x^{R}_{i} \} \) be the posterior probability of any symbol \( v_{i} \) of the codeword \( \mathbf{v} \). We find the corresponding quantity \( y^{(u)}_{i} = 2q^{(u)}_{i} - 1 \), which is (see [10, eq. (18)])

\[
y^{(u)}_{i} = y^{(u)}_{i}.
\]

Symbols \( y^{(u)}_{i} \) form the vector \( y^{(u)} \) of length \( n/2 \). Then we use some soft-decision decoder \( \Psi_{s}(y^{(u)}) \) that gives a vector \( \hat{\mathbf{y}} \in \{ m-1 \} \) and its information block \( \hat{a} \).
Step 2. Now we assume that Step 1 gives correct vector $\hat{v} = v$. Let $q^+_{u} = \Pr \{u = 1 | x^+, x''\}$ be the posterior probability of a symbol $u$. Then, the corresponding quantity $y^+_{u} = 2q^+_{u} - 1$ is (see [10, eq. (19)])

$$
y^+_u = (y^+_l + \hat{y}^+_l)/(1 + y^+_l)\hat{y}^+_l
$$

(3)

where $\hat{y}^+_l = y^+_l/\hat{v}^+_l$. The symbols $y^+_u$ form the vector $y^+$ of length $n/2$. We use some (soft-decision) decoding algorithm $P_u(y^+_u)$ to obtain a vector $\hat{u} \in \{ + \}$ and its information block $\hat{v}^+$. 

In a more general scheme $\Psi^+_{r}$, vectors $y^+$ and $y^+$ are not decoded but used as our new inputs $g$. These inputs are recalculated multiple times according to (2) and (3). Finally, we reach the end nodes $\{ s \}$ and $\{ h \}$, where we perform ML decoding as follows.

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

$$
\Pr(c_i | y_l) = (1 + c_i y_l)/2
$$
to a symbol $c_i = \pm 1$. In this way, we can find the posterior probability

$$
P(c | g) = \prod_{l=1}^{2^s}(1 + c_i y_l)/2
$$

(4)
of any codeword $c \in \{ s \}$, and choose the most probable codeword $\hat{c}$, where

$$
\forall e \in \{ s \}, \exists \hat{e} \in \{ h \} : P(\hat{e} | g) \geq P(c | g).
$$

(5)

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

Algorithm $\Psi^+_{r}$ for an input vector $g$.

1. If $0 \leq r < m$, execute the following.
   1.1. Calculate vector $y^+ = y^+ + y^+$.
   Decode $y^+$ into vector $\hat{e} \in \Psi^+_{r-1}(y^+)$. 
   Pass $\hat{e}$ and $\hat{a}$ to Step 1.2.

1.2. Calculate vector $y^+ = (y^+ + \hat{y}^+) / (1 + y^+ \hat{y}^+)$. 
   Decode $y^+$ into vector $\hat{u} \in \Psi^-_{r-1}(y^+)$. 
   Output decoded components:
   $\hat{a} := (\hat{a}^+ | \hat{a}^-) ; \quad \hat{e} := (\hat{e} \hat{u})$.
   2. If $r = 0$, use ML decoding (5) for $\{ s \}$.
   3. If $r = m$, use ML decoding (5) for $\{ h \}$.

Note that this algorithm $\Psi^+_{r}$ differs from the simplified algorithm $\Phi^+_{r}$ of [10] in three aspects. First, we use exact recalculations (3) instead of the former simplification

$$
y^+ = (y^+ + \hat{y}^+)/(1 + \hat{y}^+).
$$

(6)

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

$$
\forall e : (\hat{e}, y^+) \geq (e, y^+)
$$

Third, we employ a different rule and stop at the repetition codes $\{ s \}$ 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

$$
| \Psi^+_{r} | \leq | \Psi^+_{r-1} | + | \Psi^+_{r-1} | + 3n.
$$

Similarly to [10], this recursion gives decoding complexity

$$
| \Psi^+_{r} | \leq 6n \min(r, m-r) + n.
$$

Thus, complexity $| \Psi^+_{r} |$ has maximum order of $3n \log n$, which is twice the complexity $| \Psi^+_{r} |$ of the algorithm $\Phi^+_{r}$ of [10].

III. LIST DECODING

To enhance algorithm $\Psi^+_{r}$, we shall use some lists of $L = 2^r$ or fewer codewords obtained on any path $\xi$. This algorithm—called $\Psi^+_{l}(L)$—increases the number of operations at most $L$ times and has the overall complexity order of $L n \log n$. Given any integer parameter $L$, 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 $\Psi^+_{r}(L)$, our input consists of $L^s$ records

$$
A = (a, \rho(a), g(a)).
$$

(7)

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

$$
\rho(a) = \prod_{l=1}^{2^m - 1} \frac{1 + e_{(a)} y_l | g(a)}{2}
$$

In our list decoding, we represent the two outcomes $\hat{a}$ as the initial edges matched with their cost functions $P(\hat{a})$. Then we proceed to the next code $\{ o \}$, which corresponds to the subsequent path denoted $\xi = 2$. Given two different decoding results $\hat{e} = c_{(a)}$, our recursion (2), (3) gives two different vectors $g(\hat{a})$ 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 $L^s$ information blocks

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

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

$$
\hat{a} := \hat{a}, a_s
$$
of depth \( s \), marked by their cost functions

\[
\rho(\vec{a}):= \rho(\vec{a}) \cdot \prod_{i=1}^{2^d} \frac{1 + c_i(a_i) y_i(\vec{a})}{2}.
\]  

(8)

Step \( s \) is completed by choosing \( L^+ \) blocks with the highest cost functions \( \rho(\vec{a}) \).

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

More generally, the algorithm is executed as follows.

Algorithm \( \Psi_L^m(L) \). Input: \( L^+ \) records \( \hat{A} = (\vec{a}, \rho(\vec{a}), y(\vec{a})) \), counter \( s = 0 \).

1. If \( 0 < r < m \), for all vectors \( y(\vec{a}) \):
   1.1. Set \( y(\vec{a}) := y(\vec{a}) y(\vec{a}) \).
   Perform decoding \( \sqrt{m} \) \( (y(\vec{a})) \).
   Pass \( L \) new records \( A \) to Step 1.2.
   1.2. Set \( \hat{y}(\vec{a}) := \hat{y}(\vec{a}) + y(\vec{a}) \)
   Perform decoding \( \sqrt{m} \) \( (y(\vec{a})) \).
   Output \( L \) new records \( A \).

2. If \( r = 0 \), take both values \( a_x = 0, 1 \).
Calculate costs (8) for each \( \{ \hat{y}(\vec{a}), a_x \} \).
Choose \( L^+ \) best blocks \( \hat{A} := (\hat{a}, a_x) \).
Set \( s := s + 1 \) and return \( L^+ \) records \( A \).

3. If \( r = m \), choose \( L^+ \) best blocks \( a_x \).
Calculate costs (8) for each \( \{ \hat{y}(\vec{a}), a_x \} \).
Choose \( L^+ \) best blocks \( \hat{A} := (\hat{a}, a_x) \).
Set \( s := s + |a_x| \) and return \( L^+ \) records \( A \).

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

\[
\sigma^2_v \geq \max\{2\sigma^2, \sigma^2_v\}.
\]

The first approximation is tight for very small \( \sigma^2 \) (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 \( v \)-direction; the more so, the worse the original channel is. By contrast, the \( u \)-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 \( 0 \)—is protected the least, and then the decoding gradually improves on the subsequent paths.

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

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 \( \Psi_L^m(L) \) 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 \( \Psi_L^m(L) \), we also exhibit the actual list size \( L(\Delta) \) at which \( \Psi_L^m(L) \) approaches the optimal ML decoding within a small margin of

\[
\Delta = 0.25 \text{ dB}.
\]

This performance loss \( \Delta \) 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 \( \Psi_L^m(L) \) gives the WER \( P = 10^{-4} \);
— the complexity estimates $|\Psi^n(L)|$ 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 $\{8\}$. 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 $\Delta$. This example and other simulation results show that the algorithm $\Psi^n(L)$ 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 $\xi$ 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 $\{n\}$. Note that our decoding algorithm $\Psi^n(L)$ runs virtually unchanged on subcodes. Indeed, the single difference arises when some information block $a_\xi$ takes only one value 0 on the corresponding left node (or less than $2^k$ values on the right node). Therefore, on each step $s$, we can proceed as before, by taking only the actual blocks $a_\xi$ left at this node after expurgation.

In the algorithm $\Psi^n(L)$, this expurgation starts with the least protected information path $0^r$ that ends at the node $\{ n-r\}$. 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 $0^{r-110}$. 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 $\{8\}$ 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 $\{8\}$. Here in Table II, we also give a few list sizes $L$, the corresponding SNRs needed to reach the output WER $P = 10^{-1}$, and the complexity estimates $|\Psi^n(L)|$ 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 $\Psi^n$. For subcodes, we also approach ML decoding with the lists reduced up to 64 times relative to the original RM codes.

B. New Permutation Techniques

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

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

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

$$
\pi(i) : (i_1, \ldots, i_m) \mapsto (i_{\pi(1)}, \ldots, i_{\pi(m)}).
$$


For example, the permutation 

\[
\pi = (2, 1, 3, 4, \ldots, m)
\]

of the original 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 \(\pi\), 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 \(\pi\). We say that two permutations \(\pi\) and \(\eta\) 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 \(\Psi^r_{\text{ML}}\).

Given some positive integer \(l\) (which is smaller than the former parameter \(L\)), we then incorporate these permutations \(\pi(i)\) into the list decoding \(\Psi^r_{\text{ML}}(l)\). Namely, we form all permutations \(y_{\pi(i)}\) of the received vector \(y\) and apply algorithm \(\Psi^r_{\text{ML}}(l)\) to each vector \(y_{\pi(i)}\). 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 \(\Psi^r_{\text{ML}}(l)\). Indeed, the single vector \(y\) in our original setting (7) is replaced by \(\binom{m}{l}\) permutations \(y_{\pi(i)}\). Thus, we use parameter \(\binom{m}{l}\) in our initial setting but keep parameter \(l\) for all decoding steps. If \(l < \binom{m}{l}\), 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 \(a\). This permutation-based algorithm is called \(\Psi^r_{\text{ML}}\) below and has complexity similar to \(\Psi^r_{\text{ML}}\) 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 \(n^r/2\), 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 \(\Psi^r_{\text{ML}}(l)\) approaches the optimal ML performance even when the combined list of \(l\) most probable candidates is reduced to \(l\) times relative to the original algorithm \(\Psi^r_{\text{ML}}(L)\). 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 \(\Psi^r_{\text{ML}}(l)\) perform only within \(\Delta = 0.25\) dB away from ML decoding. These sizes and complexity estimates \(\Psi^r_{\text{ML}}(l)\) are also given in Table III. Note that both algorithms give smaller lists once this performance loss \(\Delta\) 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 \(\Psi^r_{\text{ML}}(l)\) 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 \(\Psi^r_{\text{ML}}(l)\) of complete RM codes and list decoding \(\Psi^r_{\text{ML}}(l)\) 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 $\Psi^m(l)$ Performs Within $\Delta = 0.25$ dB from ML Decoding at WER $10^{-4}$

<table>
<thead>
<tr>
<th>RM Code</th>
<th>${s}$</th>
<th>${s}$</th>
<th>${s}$</th>
<th>${s}$</th>
<th>${s}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>List size $l(\Delta)$</td>
<td>64</td>
<td>128</td>
<td>128</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>Complexity $</td>
<td>T^m(l)</td>
<td>$</td>
<td>216752</td>
<td>655805</td>
<td>777909</td>
</tr>
<tr>
<td>SNR at $10^{-4}$</td>
<td>2.91</td>
<td>2.65</td>
<td>3.38</td>
<td>5.2</td>
<td></td>
</tr>
</tbody>
</table>

TABLE IV
RM Codes of Length 256: The List Sizes, Complexities, and the Corresponding SNRs, at Which the Permutation Algorithm $\Psi^m(l)$ Performs Within $\Delta = 0.5$ dB from ML Decoding at WER $10^{-4}$

<table>
<thead>
<tr>
<th>RM Code</th>
<th>${s}$</th>
<th>${s}$</th>
<th>${s}$</th>
<th>${s}$</th>
<th>${s}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>List size $l(\Delta)$</td>
<td>32</td>
<td>64</td>
<td>64</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Complexity $</td>
<td>T^m(l)</td>
<td>$</td>
<td>116741</td>
<td>333506</td>
<td>389368</td>
</tr>
<tr>
<td>SNR at $10^{-4}$</td>
<td>3.12</td>
<td>2.82</td>
<td>3.55</td>
<td>5.4</td>
<td></td>
</tr>
</tbody>
</table>

Note, however, that the algorithm $\Psi^m(l)$ gives almost no advantage for the subcodes considered in the previous subsection. Indeed, these subcodes are obtained by eliminating the lefmost (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 $n!$ permutations of all $n$ 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 $\Psi^m(l)$;
- analyze the algorithm $\Psi^m(l)$ analytically;
- modify the algorithm $\Psi^m(l)$ 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 $\xi$. 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 $\Psi^m(L)$ and $\Psi^m(l)$.

ACKNOWLEDGMENT

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

REFERENCES


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 $[7, n/2, d]$ 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 $0 \leq 0 \leq 4$. Then $C_0$ is a subcode of codimension 1. The shadow $S$ of $C$ is defined to be $C_0 \setminus C$. 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 990-8579, 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.

0018-9448/$20.00 © 2006 IEEE