Designing Efficient Maximum-Likelihood Soft-Decision Decoding Algorithms for Linear Block Codes Using Algorithm A*

Yunghsiang S. Han
Syracuse University

Carlos R.P. Hartmann
Syracuse University, chartman@syr.edu

SU-CIS-92-10

*Designing Efficient Maximum-Likelihood Soft-Decision Decoding Algorithms for Linear Block Codes Using Algorithm A**

Yunghsiang S. Han and Carlos R.P. Hartmann

June 1992

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, NY 13244-4100

Designing Efficient Maximum-Likelihood Soft-Decision Decoding Algorithms for Linear Block Codes Using Algorithm A*

Yunghsiang S. Han[1]

Carlos R. P. Hartmann[2]

[1]Y. S. Han is with the School of Computer and Information Science at Syracuse University, Syracuse, NY 13244-4100 (e-mail: yshan@top.cis.syr.edu).

[2]C. R. P. Hartmann is with the School of Computer and Information Science at Syracuse University, Syracuse, NY 13244-4100 (e-mail: hartmann@top.cis.syr.edu).

This work used the computational facilities of the Northeast Parallel Architectures Center (NPAC) at Syracuse University.

Abstract

In this report we present a class of efficient maximum-likelihood soft-decision decoding algorithms for linear block codes. The approach used here is to convert the decoding problem into a search problem through a graph which is a trellis for an equivalent code of the transmitted code. Algorithm A^*, which uses a priority-first search strategy, is employed to search through this graph. This search is guided by an evaluation function f defined to take advantage of the information provided by the received vector and the inherent properties of the transmitted code. This function f is used to drastically reduce the search space and to make the decoding efforts of this decoding algorithm adaptable to the noise level. For example, simulation results for the (128,64) binary extended BCH code indicate that for most real channels the proposed decoding algorithm is at least fifteen orders of magnitude more efficient in time and in space than that proposed by Wolf. Simulation results for the (104, 52) binary extended quadratic residue code are also given. These simulation results indicate that the use of Algorithm A^* for decoding has resulted not only in an efficient soft-decision decoding algorithm for hitherto intractable linear block codes, but an algorithm which is in fact optimal as well.

1 Introduction

Block codes and convolutional codes are two well-known error-control techniques for reliable transmission of digital information over noisy communication channels. Linear block codes with coding gains far superior to those of convolutional codes have been known for many years. However, these block codes have not been used in practice for lack of an efficient soft-decision decoding algorithm.

This report deals with the *maximum-likelihood soft-decision decoding of linear block codes.* By *maximum-likelihood decoding*, we mean a decoding algorithm which minimizes the probability of decoding to an incorrect codeword when all codewords have equal probability of being transmitted. By *soft-decision* we mean that the decoding algorithm can use real numbers (e.g., the analog output of filters matched to the signals) associated with every component of the codeword in the decoding procedure. Soft-decision decoding can provide about 2 dB of additional coding gain when compared to hard-decision decoding.

Our approach to the maximum-likelihood soft-decision decoding of linear block codes is to convert this problem into a search problem through a graph which is a trellis for a code equivalent to the transmitted code. In [12] a novel maximum-likelihood soft-decision decoding algorithm that is applicable to any linear block code, and uses Algorithm A^* to search through this graph, is proposed. Algorithm A^*, which uses a priority-first search strategy, is widely used in Artificial Intelligence search problems [20]. This search is guided by an evaluation function f defined to take advantage of the information provided by the received vector and the inherent properties of the transmitted code. This function f is used to drastically reduce the search space and to make the decoding efforts of this decoding algorithm adaptable to the noise level.

In this report we introduce a new class of f functions. Simulation results for the (104,52) binary extended quadratic residue code, and the (128,64) binary extended BCH code attested to the fact that, in general, the decoding algorithm that uses an f function from this class is at least one order of magnitude more efficient, in time and space, than the decoding algorithm that uses the f function defined in [12].

In Section 2 we review maximum-likelihood decoding of linear block codes and describe Algorithm A^*. In the next section we present our new decoding algorithm and give the simulation results for the (104,52) and the (128,64) codes. Concluding remarks are presented in Section 4.

2 Preliminaries

Let $\boldsymbol{V}_{n,q}$ be the set of all n-tuples over $\boldsymbol{GF(q)}$. A q-ary (n,k) linear block code $\boldsymbol{C}$ is a subspace of $\boldsymbol{V}_{n,q}$ of dimension k. $\boldsymbol{C}$ can be characterized by a generator matrix $\boldsymbol{G}$ or by a parity-check matrix $\boldsymbol{H}$. Any set of k linearly independent vectors in $\boldsymbol{C}$ can be used as the rows of $\boldsymbol{G}$. On the other hand, any set of $n-k$ linearly independent vectors in $\boldsymbol{C}^{\perp}$ (null space of C) can be used as the rows of $\boldsymbol{H}$. Thus, a vector in $\boldsymbol{V}_{n,q}$ is a codeword in $\boldsymbol{C}$ if and only if it is a linear combination over $\boldsymbol{GF(q)}$ of the rows of $\boldsymbol{G}$. Therefore, a codeword in $\boldsymbol{C}$ can be written as $\boldsymbol{c} = \boldsymbol{u} \cdot \boldsymbol{G}$ where $\boldsymbol{u}$ is a k-tuple over $\boldsymbol{GF(q)}$. Since $\boldsymbol{C}$ is the null space of $\boldsymbol{C}^{\perp}$, any vector $\boldsymbol{v} \in \boldsymbol{V}_{n,q}$ is a codeword of $\boldsymbol{C}$ iff it is orthogonal to every row of $\boldsymbol{H}$, that is, $\boldsymbol{v} \cdot \boldsymbol{H}^T = \boldsymbol{0}$.

Let $\boldsymbol{c} = (c_0, c_1, \ldots, c_{n-1})$, $c_j \in \boldsymbol{GF(q)}$ be a codeword of $\boldsymbol{C}$ transmitted over a time-discrete memoryless channel with output alphabet $\boldsymbol{B}$. Furthermore, let $\boldsymbol{r} = (r_0, r_1, \ldots, r_{n-1})$, $r_j \in \boldsymbol{B}$ denote the received vector, and assume that $\boldsymbol{Pr}(r_j|c_i) > 0$ for $r_j \in \boldsymbol{B}$ and $c_i \in \boldsymbol{GF(q)}$. Let $\hat{\boldsymbol{c}}$ be an estimate of the transmitted codeword $\boldsymbol{c}$.

The *maximum-likelihood decoding rule* (**MLD** rule) for a time-discrete memoryless channel can be formulated as:

$$\text{set } \hat{\boldsymbol{c}} = \boldsymbol{c}_\ell \text{ where } \boldsymbol{c}_\ell = (c_{\ell 0}, c_{\ell 1}, \ldots, c_{\ell(n-1)}) \in \boldsymbol{C} \text{ and}$$

$$\prod_{j=0}^{n-1} \boldsymbol{Pr}(r_j|c_{\ell j}) \geq \prod_{j=0}^{n-1} \boldsymbol{Pr}(r_j|c_{ij}) \text{ for all } \boldsymbol{c}_i = (c_{i0}, c_{i1}, \ldots, c_{i(n-1)}) \in \boldsymbol{C}.$$

Let $S(\boldsymbol{c}_i, \boldsymbol{c}_\ell) \subseteq \{0, 1, \ldots, n-1\}$ be defined as $j \in S(\boldsymbol{c}_i, \boldsymbol{c}_\ell)$ iff $c_{\ell j} \neq c_{ij}$. Then the **MLD** rule can be written as

$$\text{set } \hat{\boldsymbol{c}} = \boldsymbol{c}_\ell \text{ where } \boldsymbol{c}_\ell \in \boldsymbol{C} \text{ and}$$

$$\sum_{j \in S(\boldsymbol{c}_i, \boldsymbol{c}_\ell)} \ln \frac{\boldsymbol{Pr}(r_j|c_{\ell j})}{\boldsymbol{Pr}(r_j|c_{ij})} \geq 0 \text{ for all } \boldsymbol{c}_i \in \boldsymbol{C}.$$

For the binary case, following the formulation given in [13], we define the bit log-likelihood ratio of r_i as

$$\phi_i = \ln \frac{\boldsymbol{Pr}(r_j|0)}{\boldsymbol{Pr}(r_j|1)}\ .$$

Furthermore, let $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_{n-1})$. By [13, Theorem 5] the **MLD** rule can be written as

$$\text{set } \hat{\boldsymbol{c}} = \boldsymbol{c}_\ell, \text{ where } \boldsymbol{c}_\ell \in \boldsymbol{C} \text{ and}$$
$$\sum_{j=0}^{n-1} (\phi_j - (-1)^{c_{\ell j}})^2 \leq \sum_{j=0}^{n-1} (\phi_j - (-1)^{c_{ij}})^2 \text{ for all } \boldsymbol{c}_i \in \boldsymbol{C}. \qquad (1)$$

In the special case where the codewords of $\boldsymbol{C}$ have equal probability of being transmitted, the **MLD** rule minimizes the error probability.

One way to implement the **MLD** rule is to calculate $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_i) = \prod_{j=0}^{n-1} \boldsymbol{Pr}(r_j|c_{ij})$ for every codeword in $\boldsymbol{C}$ and select the codeword that maximizes it. In practice this can be done only for those codes with a small number of codewords, that is, low rate codes or middle-to-high rate codes with short block length.

However, in 1979 Hwang [13] showed that it is possible to select a codeword maximizing $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_i)$ without calculating it for every codeword in a binary code $\boldsymbol{C}$. He proved that if a codeword $\boldsymbol{c}_{i1} = \boldsymbol{c}_{i2} \oplus \boldsymbol{c}_{i3}$ where $\boldsymbol{c}_{i2}$ and $\boldsymbol{c}_{i3}$ are disjoint codewords, then we can select a codeword maximizing $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_i)$ without directly calculating $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_{i1})$. Recently it has been shown that if the **MLD** rule is implemented using Hwang's technique, then the codewords that can be dropped from the computation are only those satisfying the above property [16, 19]. In 1980 Hwang [14] proposed another approach to reduce the number of codewords that need to be considered when applying the MLD rule. However, since the k most "reliable" positions of the received vector may not be linearly independent, it is simple to design an example where the procedure proposed in [14] will fail to start. Such an example is given in Appendix A.

Several researchers [6, 25, 22] have presented a technique for decoding linear block codes that converts the decoding problem into a graph-search problem on a trellis derived from the parity-check matrix of the code. Thus the MLD rule can be implemented by applying the Viterbi Algorithm [24] to this trellis. Therefore, in practice this breadth-first search scheme can be applied only to codes with small redundancy, that is, small $n - k$ or codes with a small number of codewords.

When the decoding problem is converted into a graph-search problem, we are interested in finding a path from the start node representing the initial condition to a goal node that represents the termination condition. This path will optimize some criterion that leads us to construct a codeword that maximizes $\boldsymbol{Pr}(\boldsymbol{r}|\boldsymbol{c}_i)$, where $\boldsymbol{c}_i \in \boldsymbol{C}$.

Thus, the decoding problem has been mapped to a more general graph-search problem. In this graph each arc is assigned a cost and the cost of a path is the sum of the costs of the arcs connecting the nodes in this path. The problem is how to find an optimal path from the start node to a goal node, that is, a path with minimal (maximal) cost. The algorithm A^*, widely used in Artificial Intelligence, is an efficient procedure for finding an optimal path if one exists in a graph.

In order to more easily describe Algorithm A^*, we first give a general graph-searching procedure as presented in [20]:

Procedure **GRAPHSEARCH**

1. Create a *search graph*, $\mathcal{G}$, consisting solely of the start node, s. Put s on a list called *OPEN*.

2. Create a list called *CLOSED* that is initially empty.

3. LOOP: if *OPEN* is empty, exit with failure.

4. Select the first node on *OPEN*, remove it from *OPEN*, and put it on *CLOSED*. Call this node m.

5. If m is a goal node, exit successfully with the solution obtained by tracing a path along the pointers from m to s in $\mathcal{G}$. (Pointers are established in Step 7.)

6. Expand node m, generating the set, M, of its successors that are not ancestors of m. Install these members of M as successors of m in $\mathcal{G}$.

7. Establish a pointer to m from those members of M that were not already in $\mathcal{G}$ (i.e., not already on either *OPEN* or *CLOSED*).Add these members of M to *OPEN*. For each member of M that was already on *OPEN* or *CLOSED*, decide whether or not to redirect its pointer to m. For each member of M already on *CLOSED*, decide for each of its descendants in $\mathcal{G}$ whether or not to redirect its pointer.

8. Reorder the list *OPEN*, either according to some arbitrary scheme or according to heuristic merit.

9. Go LOOP.

If the graph being searched is not a tree, it is possible that some of the elements of set M have already been visited—that is, they are already on list OPEN or list CLOSED. The problem of determining whether a newly generated node is on these lists can be computationally very expensive. For this reason we may decide to avoid making this test, in which case the search tree may contain several repeated nodes. These node repetitions lead to redundant successor computations and there is a trade-off between the computation cost for testing for repeated nodes and the computation cost for generating a larger search tree. In steps 6 and 7 of procedure **GRAPHSEARCH**, testing for repeated nodes is performed.

In an uninformed search procedure no heuristic information from the problem has been used in reordering the list OPEN in Step 8. In this case, the two well-known search methods are the breadth-first and depth-first. However, these methods are exhaustive in nature, and thus in practice are applicable only to graphs with small numbers of nodes and paths.

In many cases it is possible to use some inherent properties of a problem to help reduce the search. The search procedure using this information is called a *heuristic search method.* In many situations it is possible to specify heuristics that reduce considerably the search effort without compromising the optimality of the solution.

One of the well-known heuristic search methods that guarantee to find the optimal solution if one exists is the Algorithm A^* [20]. The description of A^* given here is taken from [20]. A^* uses a cost function called *evaluation function* f to guide the search through the graph. This function f is computed for every node that is added to list OPEN in Step 7 of the procedure **GRAPHSEARCH**. In Step 8 of this procedure, we reorder the list OPEN according to the value of the function f. From now on, in order to simplify the description of A^*, we assume that an optimal path is one that minimizes the cost function.

For every node m, we define the evaluation function f so that its value $f(m)$ at node m estimates the cost of the minimum cost path that goes through node m. $f(m)$ is computed as

$$f(m) = g(m) + h(m),$$

where $g(m)$ estimates the cost of the minimal cost path from the start node s to node m, and $h(m)$ estimates the cost of the minimal cost path from node m to a goal node. We call h the *heuristic function.*

In A^*, the next node to be expanded is the one with the smallest value of f on the list OPEN since this node imposes the least severe constraints.

Similarly, let f^* be a function such that $f^*(m)$ at any node m is the actual cost of a minimum cost path that goes through node m. Analogously,

$$f^*(m) = g^*(m) + h^*(m),$$

where $g^*(m)$ is the actual cost of a minimum cost path from the start node s to node m, and $h^*(m)$ is the actual cost of a minimum cost path from node m to a goal node.

A^* requires that $g(m) \geq g^*(m)$ and $h(m) \leq h^*(m)$ for every node m of the graph. These requirements guarantee that A^* will find a minimum cost path if one exists; however, if the graph is finite, then the only condition that must be satisfied to guarantee optimality is $h(m) \leq h^*(m)$ for every node m of the graph [20].

An obvious choice for $g(m)$ is the cost of the path in the search tree from the start node s to node m given by summing all the arc costs encountered while constructing the minimum cost path from the start node s to node m. Note that this path is the lowest cost path from the start node s to node m found so far by the algorithm. The value of $g(m)$ may decrease if the search tree is altered in Step 7 of procedure **GRAPHSEARCH**. From now on we assume that function g is calculated in this way. In this case $g(m) \geq g^*(m)$ for every node m of the graph. Furthermore, if $h(m) = 0$ for any node m, then A^* becomes a version of Dijkstra's algorithm [9].

To define $h(m) \leq h^*(m)$, we use the properties of the problem. It can be shown [20] that if we have two evaluation functions $f_1(m) = g_1(m) + h_1(m)$ and $f_2(m) = g_2(m) + h_2(m)$ satisfying $h_1(m) < h_2(m) \leq h^*(m)$ for every node m, the A^* using evaluation function f_2 will never expand more nodes than the A^* using evaluation function f_1. Furthermore, if there exists a unique optimal path, then the above results hold when $h_1(m) \leq h_2(m) \leq h^*(m)$ is satisfied for every node m. Also, if $h(m) > 0$ for any node m, then A^*, using this function h, will never expand more nodes than the above version of Dijkstra's algorithm.

The monotone restriction is a reasonable restriction that when imposed on h can substantially decrease the computation time and storage of A^*. In [20], the function h is said to satisfy the monotone restriction if and only if for all nodes m_i and m_j, such that node m_j is a successor of node m_i,

$$0 \leq h(m_i) - h(m_j) \leq c(m_i, m_j)$$

with $h(t) = 0$, where t is any goal node and $c(m_i, m_j)$ is the arc cost between node m_i and node m_j.

If the monotone restriction is satisfied, then it can be shown [20] that A^* has already found an optimal path from the start node to the node it selects to expand. Thus there is no need for A^* to check if the newly generated nodes are on the list CLOSED and we do not have to store this list. Furthermore, we do not have to update the parentage in the search tree of any successors of the node A^* selects to expand. Also, if the monotone restriction is satisfied, the f values of the sequence of nodes expanded by A^* is nondecreasing [20].

In the proof of the above results [20], the conditions that are imposed by the monotone restriction, namely $0 \leq h(m_i) - h(m_j)$ and $h(t) = 0$, have not been used. So the only requirement for this proof is that

$$h(m_i) \leq h(m_j) + c(m_i, m_j). \tag{2}$$

We will show that the h function we use in the next section will satisfy this inequality, so we can still use the above result to speed up the decoding procedure. It is easy to find an example to show that this h function does not satisfy $0 \leq h(m_i) - h(m_j)$.

Another property of A^* [20, Prob. 2.6] that will be used in our decoding algorithm is as follows: Algorithm A^* still finds the optimal path (if one exists) if it removes from list OPEN any node m for which $f(m) > UB$, where UB is an upper bound on the cost of an optimal path.

From the description of A^* it is clear that the most important factor in the efficiency of A^* is the selection of the heuristic function h and, consequently, the evaluation function f.

3 Decoding algorithm

For ease of explanation we will assume from now on that the received vector is $\boldsymbol{\phi}$ instead of $\boldsymbol{r}$.

Our decoding algorithm, guided by an evaluation function f, searches through a graph that is a trellis for a code $\boldsymbol{C}^*$, which is equivalent to code $\boldsymbol{C}$. $\boldsymbol{C}^*$ is obtained from $\boldsymbol{C}$ by permuting the positions of codewords of $\boldsymbol{C}$ in such a way that the first k positions of codewords in $\boldsymbol{C}^*$ correspond to the "most reliable linearly independent" positions in the received vector $\boldsymbol{\phi}$. In Appendix B we give an algorithm to obtain $\boldsymbol{G}^*$ from $\boldsymbol{G}$. $\boldsymbol{G}^*$ is a generator matrix of $\boldsymbol{C}^*$ whose first k columns form the $k \times k$ identity matrix. The time complexity of this algorithm is also discussed in this appendix.

In our decoding algorithm the vector $\boldsymbol{\phi}^* = (\phi_0^*, \phi_1^*, \ldots, \phi_{n-1}^*)$ is used as the "received vector." It is obtained by permuting the positions of $\boldsymbol{\phi}$ in the same manner in which the columns of $\boldsymbol{G}$ can be permuted to obtain $\boldsymbol{G}^*$.

3.1 Construction of trellis

We now give a short description of a trellis [1] for the code $\boldsymbol{C}^*$ where the search will be performed. We remark here that even though we will describe the complete trellis, our decoding algorithm will construct only a very small subgraph of this trellis during the decoding procedure. Let $\boldsymbol{H}^*$ be a parity-check matrix of $\boldsymbol{C}^*$, and let $\boldsymbol{h}_i{}^*$, $0 \leq i < n$ be the column vectors of $\boldsymbol{H}^*$. Furthermore, let $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*)$ be a codeword of $\boldsymbol{C}^*$. With respect to this codeword, we recursively define the states $\boldsymbol{s}_t$, $-1 \leq t < n$, as follows:

$$\boldsymbol{s}_{-1} = \boldsymbol{0}$$

and

$$\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t^* \boldsymbol{h}_t^* = \sum_{i=0}^{t} c_i^* \boldsymbol{h}_i^*, \quad 0 \leq t < n.$$

Clearly, $\boldsymbol{s}_{n-1} = \boldsymbol{0}$ for all codewords of $\boldsymbol{C}^*$. The above recursive equation can be used to draw a trellis diagram. In this trellis, $\boldsymbol{s}_{-1} = \boldsymbol{0}$ identifies the start node which is at level -1; $\boldsymbol{s}_{n-1} = \boldsymbol{0}$ identifies the goal node which is at level $n-1$; and each state $\boldsymbol{s}_t, 0 \leq t < n-1$ identifies a node at level t. Furthermore, each transition (arc) is labelled with the appropriate codeword bit c_t^*. A more detailed description of a trellis for a linear block code can be found in [25]. Note that the trellis defined here corresponds to the expurgated trellis of [25].

3.2 Evaluation function

As we pointed out before, the selection of evaluation function f is of the utmost importance, since it determines the search effort of A^*. We now describe the function f we use in our decoding algorithm.

In order to define function f, we need first to specify the arc costs. In the trellis of $\boldsymbol{C}^*$, the cost of the arc from $\boldsymbol{s}_{t-1}$ to $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + c_t^* \boldsymbol{h}^*{}_t$ is assigned the value $(\phi_t^* - (-1)^{c_t^*})^2$. Thus the solution of the decoding problem is converted into finding a path from the start node to the goal node, that is, a codeword $\boldsymbol{c}^* = (c_0^*, c_1^*, \ldots, c_{n-1}^*)$ such that $\sum_{i=0}^{n-1} (\phi_i^* - (-1)^{c_i^*})^2$ is minimum among all paths from the start node to the goal node.

Now we define function f for every node m in the trellis as follows:

$$f(m) = g(m) + h(m).$$

As previously noted, $g(m)$ is the lowest cost path from the start node to node m found so far by the algorithm, where the cost of a path from the start node to node m is obtained by summing all the arc costs encountered while constructing this path.

We now define a class of heuristic functions. Furthermore, if a function h belongs to this class it will satisfy $h(m) \leq h^*(m)$ for every node m. Recall that $h^*(m)$ is the cost of a minimum cost path from node m to the goal node. In order to define a function h which is a "good" estimator of h^* we must use properties of the linear block code which are invariant under any permutation of the positions of the codewords.

Let $HW = \{w_i | 0 \leq i \leq I\}$ be the set of all distinct Hamming weights that codewords of $\boldsymbol{C}$ may have. Furthermore, assume $w_0 < w_1 < \cdots < w_I$. Our heuristic functions are defined to take into consideration the fact that the Hamming distance between any two codewords of $\boldsymbol{C}^*$ must belong to HW, and the linear property of $\boldsymbol{C}^*$.

Let S_{C^*} be a given subset of $\boldsymbol{C}^*$, and $P_i(S_{C^*})$ be the set that contains all the subsets of S_{C^*} of cardinality $i, 0 \leq i \leq |S_{C^*}|$, where $|S_{C^*}|$ is the cardinality of S_{C^*}. For a given S_{C^*} we now define our heuristic function, $h^{(i)}$, of order $i, 0 \leq i \leq |S_{C^*}|$.

1. For nodes at level $\ell, -1 \leq \ell < k-1$:

   Let m be a node at level ℓ, and $v_0, v_1, \ldots, v_\ell$ be the labels of the lowest cost path $\boldsymbol{P}'_m$ from the start node to node m found so far by the algorithm. Furthermore, let $\boldsymbol{v}_t = (v_{\ell+1}, v_{\ell+2}, \ldots, v_{n-1})$ be a binary $(n-\ell-1)$-tuple and $\boldsymbol{v} = (v_0, v_1, \ldots, v_\ell, v_{\ell+1}, v_{\ell+2}, \ldots, v_{n-1})$. Denote by $d_H(\boldsymbol{x},\boldsymbol{y})$ the Hamming distance between $\boldsymbol{x}$ and $\boldsymbol{y}$, and by $W_H(\boldsymbol{x}) = d_H(\boldsymbol{x},\boldsymbol{0})$.

   If $S_{C^*} = \emptyset$, then $h^{(0)}(m) = 0$. Otherwise, let $Y_i \in P_i(S_{C^*})$, and

   $$T(m, Y_i) = \{\boldsymbol{v}_t | \forall \boldsymbol{c}^* \in Y_i, d_H(\boldsymbol{v},\boldsymbol{c}^*) \in HW\}.$$

   Note that $0 \leq W_H(\boldsymbol{v}_t) \leq (n-\ell-1)$ for all $\boldsymbol{v}_t \in T(m, Y_i)$. Also note that $T(m, Y_i) \neq \emptyset$ for any $Y_i \in P_i(S_{C^*})$. This can easily be seen by constructing a $\boldsymbol{v}_t \in T(m, Y_i)$ as follows. Consider the binary n-tuple $\boldsymbol{u} \cdot \boldsymbol{G}^* = (c_0^*, c_1^*, \ldots, c_\ell^*, c_{\ell+1}^* \ldots, c_{n-1}^*)$, where $\boldsymbol{u} = (v_0, v_1, \ldots, v_\ell, 0, \ldots, 0)$ is a binary k-tuple. Clearly, $c_i^* = v_i$ for $0 \leq i \leq \ell$. Thus, $\boldsymbol{v}_t = (c_{\ell+1}^*, c_{\ell+2}^*, \ldots, c_{n-1}^*) \in T(m, Y_i)$.

Finally, we define $h^{(i)}$ as

$$h^{(i)}(m) = \max_{Y_i \in P_i(S_{C^*})} \left\{ \min_{\boldsymbol{v}_t \in T(m,Y_i)} \left\{ \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \right\} \right\}.$$

2. For nodes at level $\ell, k-1 \leq \ell < n$:

   Because of the linear property of $\boldsymbol{C}^*$ and the fact that the first k columns of $\boldsymbol{G}^*$ are linearly independent, there is only one path from any node at level $k-1$ to the goal node. Furthermore, we can easily determine the labels $v_k^*, v_{k+1}^*, \ldots, v_{n-1}^*$ of this path using $\boldsymbol{G}^*$ and calculate its cost $\sum_{i=k}^{n-1} \left(\phi_i^* - (-1)^{v_i^*}\right)^2$. In view of the above fact, we define function $h^{(i)}$ as follows:

$$h^{(i)}(m) = \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v_i^*}\right)^2,$$

   where $v_{\ell+1}^*, v_{\ell+2}^*, \ldots, v_{n-1}^*$ are the labels of the only path $\boldsymbol{P}_m$ from node m to the goal node.

   Note that if node m is the goal node, then $h^{(i)}(m) = 0$. Furthermore, $h^{(i)}(m) = h^*(m)$ since there is only one path from node m to the goal node and $h^{(i)}(m)$ is the cost of this path.

Obviously, $h^{(i)}(m) \leq h^*(m)$ for any node m in the trellis.

For a given S_{C^*} and i, the evaluation function f is $f(m) = g(m) + h^{(i)}(m)$.

It is very important that the time complexity for calculating $h^{(i)}(m)$ be "reasonable," for otherwise the time taken by the decoding algorithm is spent calculating $h^{(i)}(m)$, even though there are only a few nodes to be visited (open) in the trellis. In Appendix C we present an algorithm to calculate $h^{(1)}(m)$ for node m at level $\ell, -1 \leq \ell < k-1$ whose time complexity is $O(|S_{C^*}| \times n)$.

We now give properties of heuristic function $h^{(i)}$ that will be used to speed up the decoding procedure. The proofs of properties 1 and 2 are given in Appendix D. Properties 3 and 4 are immediate consequences of the definition of function $h^{(i)}$.

Property 1. For a given S_{C^*}

$$h^{(i)}(m_j) \leq h^{(i)}(m_\ell) + c(m_j, m_\ell),$$

where node m_ℓ is an immediate successor of node m_j, and $c(m_j, m_\ell)$ is our arc cost from node m_j to node m_ℓ.

Property 2. For a given S_{C^*} and i, if nodes $m_{\ell 1}$ and $m_{\ell 2}$ are immediate successors of node m_j, then

$$f(m_{\ell 1}) = f(m_j) \quad \text{or} \quad f(m_{\ell 2}) = f(m_j).$$

Now let S_{C^*} and S'_{C^*} be nonempty subsets of $\boldsymbol{C}^*$, and $h^{(i)}(h'^{(i)})$ be the i^{th} order heuristic function corresponding to $S_{C^*}(S'_{C^*})$.

Property 3. If $S_{C^*} \subseteq S'_{C^*}$ and $0 \leq i \leq |S_{C^*}|$, then

$$h^{(i)}(m) \leq h'^{(i)}(m) \text{ for every node } m.$$

Property 4. If $0 \leq i \leq j \leq |S_{C^*}|$, then

$$h^{(i)}(m) \leq h^{(j)}(m) \text{ for every node } m.$$

We remark here that the decoding algorithm using function $h^{(j)}$ $\left(h'^{(i)}\right)$ will not open and store more nodes than the decoding algorithm using function $h^{(i)}$ $\left(h^{(i)}\right)$. However, the time complexity for calculating $h^{(j)}(m)$ $\left(h'^{(i)}(m)\right)$ will be higher than that of $h^{(i)}(m)$ $\left(h^{(i)}(m)\right)$.

For the special case of $S_{C^*} = \{\mathbf{0}\}$, the first-order heuristic function is the heuristic function proposed in [12].

When a first-order heuristic function $h^{(1)}$ is used, the time and space complexities of the algorithm proposed here are $O(|S_{C^*}| \times n \times N(\boldsymbol{r}))$ and $O(n \times M(\boldsymbol{r}))$, respectively, where

$$\begin{aligned} N(\boldsymbol{r}) &= \text{the number of nodes visited during the decoding of } \boldsymbol{r}, \\ M(\boldsymbol{r}) &= \text{the maximum number of nodes that need to be stored} \\ &\quad\ \text{during the decoding of } \boldsymbol{r}. \end{aligned}$$

The derivation of these results is similar to that of the time and space complexities given in [12] for the algorithm proposed there.

For long block codes it may be impossible to determine the set HW. However, our algorithm will still find the optimal solution even if in the computation of function h the algorithm considers all the Hamming weights of any superset of HW. The algorithm using a superset of HW may visit more nodes than that using HW. Furthermore, in most cases the received vector is closed to a unique codeword. In this case, as pointed out in Section 2, the algorithm will not open fewer nodes if it uses a proper superset of HW instead of HW in the computation of heuristic function.

3.3 Speed-up techniques

In this subsection we present some properties of the decoding algorithm that can be used to speed up the decoding procedure. In order to simplify the presentation of these techniques we assume that function h belongs to the class of heuristic functions defined above.

By Property 1, function h satisfies the property,

$$h(m_i) \leq h(m_j) + c(m_i, m_j),$$

where node m_j is an immediate successor of node m_i and $c(m_i, m_j)$ is our arc cost from node m_i to node m_j. Then, as we pointed out before, we do not need to store the list CLOSED and we do not have to update the parentage in the search tree of any successors of the node that our algorithm selects to expand.

By Property 2, when our algorithm expands a node m at level $\ell < k-2$, we need to compute the value of function f for only one of its successors. This is because the value of function f for the other successor is equal to that of node m and we can easily determine which successor has the value $f(m)$. Thus our algorithm is a depth-first search type Algorithm A^*.

Furthermore, since our function h satisfies Inequality 2, by the remark in the previous section, the f values of the sequence of nodes expanded by our algorithm is nondecreasing. Let node m_1 at level $\ell < k-2$ be the first node of list OPEN. Consider the sequence of nodes that the algorithm will follow from node m_1 to node m_2 which is at level $k-2$. Due to the above properties, the value of the function f at every one of these nodes is equal to $f(m_1)$. Furthermore, the labels of the path corresponding to this sequence of nodes can be easily determined by the first $k-\ell-2$ positions of the binary $(n-\ell-1)$-tuple $(v'_{\ell+1},\ldots,v'_{k-2},v'_{k-1},\ldots,v'_{n-1})$ used to calculate $h(m_1)=\sum_{i=\ell+1}^{n-1}\left(\phi_i^*-(-1)^{v'_i}\right)^2$. Thus, we do not have to visit the nodes of this sequence. This reduces considerably the total number of nodes visited.

Our algorithm will search the trellis only up to level $k-1$ since we can construct the only path from any node m at level $k-1$ to the goal node using $\boldsymbol{G}^*$. The labels of the combined paths from the start node to node m, and from node m to the goal node, correspond to a codeword. So the cost of this path, which is equal to $f(m)$, can be used as an upper bound on the cost of an optimal path. As noted in Section 2, we can use this upper bound to reduce the size of list OPEN. Furthermore, since there is a codeword whose corresponding path in the trellis has cost equal to $f(m)$, then we need to keep only one node on list OPEN whose f value is equal to the upper bound.

The trellis search can be stopped at any time when we know that a codeword $\boldsymbol{c}_\ell^*=\left(c_{\ell 0}^*,c_{\ell 1}^*,\ldots,c_{\ell(n-1)}^*\right)$ generated satisfies Inequality 1. The following criterion can be used to indicate this fact.

Criterion. If $h^{(i)}(\boldsymbol{s}_{-1})=\sum_{j=0}^{n-1}\left(\phi_j^*-(-1)^{c_{\ell j}^*}\right)^2$, then $\boldsymbol{c}_\ell^*$ satisfies Inequality 1.

Recall that $\boldsymbol{s}_{-1}$ is the start node.

The validity of this criterion is based on the fact that, since $\boldsymbol{C}^*\subseteq T(\boldsymbol{s}_{-1},Y_i)$, then $h^{(i)}(\boldsymbol{s}_{-1})\le\sum_{j=0}^{n-1}\left(\phi_j^*-(-1)^{c_{\iota j}^*}\right)^2$ for any $\boldsymbol{c}_\iota^*\in\boldsymbol{C}^*$.

Note that the decision criterion introduced in [23] is equivalent to the above criterion for the special case, $S_{C^*}=\left\{\boldsymbol{c}_\ell^*\right\}$. It is easy to show that if a codeword $\boldsymbol{c}_\ell^*$ satisfies the criterion given by Inequalities 3.7a and 3.7b in [11], then it will also satisfy the criterion given in [23].

It is important to mention that the set S_{C^*} does not need to be fixed during the decoding of $\boldsymbol{\phi}$. In the case where S_{C^*} is allowed to change, we have an adaptive decoding procedure. However, we cannot any longer guarantee that Inequality 2 will be satisfied.

4 Simulation results for the AWGN channel

In this section we present simulation results for the $(104, 52)$ binary extended quadratic residue code and the $(128, 64)$ binary extended BCH code when these codes are transmitted over the Additive White Gaussian Noise (AWGN) channel. We assume that antipodal signaling is used in the transmission so that the j^{th} components of the transmitted codeword $\boldsymbol{c}$ and received vector $\boldsymbol{r}$ are

$$c_j = (-1)^{c_j}\sqrt{E} \quad \text{and} \quad r_j = (-1)^{c_j}\sqrt{E} + e_j,$$

respectively, where E is the signal energy per channel bit and e_j is a noise sample of a Gaussian process with single-sided noise power per hertz N_0. The variance of e_j is $N_0/2$ and the SNR for the channel is $\gamma = E/N_0$. In order to account for the redundancy in codes of different rates, we used the SNR per transmitted information bit $\gamma_b = E_b/N_0 = \gamma n/k$ in our simulation.

We do not know HW for these two codes, so we use a superset for them. For (104,52) we know that $d_{\min} = 20$ and that the Hamming weight of any codeword is divisible by 4 [17]. Thus, for this code the superset used is $\{x | (x \text{ is divisible by 4 and } 20 \leq x \leq 84) \text{ or } (x = 0) \text{ or } (x = 104)\}$. For (128,64), the superset used is $\{x | \ (x \text{ is even and } 22 \leq x \leq 106) \text{ or } (x = 0) \text{ or } (x = 128)\}$.

We have implemented our adaptive decoding algorithm for the case $i = 1$, that is, we use a first-order heuristic function. Furthermore, the set S_{C^*} has cardinality 1 and is updated according to the following rule: for every codeword $\mathbf{c^*}_1$ generated during the decoding of ϕ, if the value of $h^{(1)}(\boldsymbol{s}_{-1})$ calculated with respect to $\mathbf{c^*}_1$ is greater than the value of $h^{(1)}(\boldsymbol{s}_{-1})$ calculated with respect to the codeword in S_{C^*}, then set $S_{C^*} = \{\mathbf{c^*}_1\}$. The rationality behind this rule is that, for any node m, $h^{(1)}(m) \geq h^{(1)}(\boldsymbol{s}_{-1})$ whenever these values are calculated with respect to the same set S_{C^*}.

Simulation results attested to the fact that the efficiency of this decoding algorithm depends strongly on the selection of the initial set S_{C^*}.

In our implementation this initial set is constructed by considering the codeword $\boldsymbol{c}^*$

obtained as follows. Let $\boldsymbol{u} = (u_0, u_1, \ldots, u_{k-1})$ where

$$u_i = \begin{cases} 0 & \text{if } \phi_i^* \geq 0; \\ 1 & \text{if } \phi_i^* < 0; \end{cases},$$

and $\boldsymbol{\phi}^* = (\phi_0^*, \phi_1^*, \ldots, \phi_{k-1}^*, \phi_k^*, \ldots, \phi_{n-1}^*)$. Now we let $S_{C^*} = \{\boldsymbol{c}^*\}$, where $\boldsymbol{c}^* = \boldsymbol{u} \cdot \boldsymbol{G}^*$.

In the implementation of our decoding algorithm we have decided not to check for repeated nodes. In this situation the graph becomes a decision tree. Thus, we do not have to keep list CLOSED. Furthermore, list OPEN is always kept ordered according to the values f of its nodes. In this case, the time complexity and the space complexity of our algorithm are $O(n \times N(\boldsymbol{r}))$ and $O(n \times M(\boldsymbol{r}))$, respectively. Recall that

$$\begin{aligned} N(\boldsymbol{r}) &= \text{the number of nodes visited during the decoding of } \boldsymbol{r}; \\ M(\boldsymbol{r}) &= \text{maximum number of nodes stored on list OPEN during the decoding of } \boldsymbol{r}. \end{aligned}$$

The values of $N(\boldsymbol{r})$ and $M(\boldsymbol{r})$ will strongly depend upon the signal to noise ratio (SNR). Up to now we do not have a "good" estimator of these values; however, they are upperbounded by $2^{k+1} - 1$. So, in the worst case, the time and space complexities of our algorithm are $O(n \times 2^k)$, which are, under the condition $k \leq (n - k)$, equal to those of Wolf's algorithm [25], which are $O(n \times \min(2^k, 2^{n-k}))$ [8].

First, we give simulation results for the (104,52) code. Quadratic residue codes are known to be very good codes that are very difficult to decode even when only hard-decision decoding is employed [4, 7, 5, 21]. Some quadratic residue codes have been decoded by using information-set decoding algorithms [3]. However, these algorithms are sub-optimal, that is, do not implement the **MLD** rule. Thus, the only two maximum-likelihood soft-decision decoding algorithms known to us that can be used to decode the (104,52) code are Wolf's algorithm [25] and Hwang's algorithm [13].

It is very hard for us to compare the performance of our algorithm with that of Hwang because he found the subset of codewords that must be stored for implementing the **MLD** rule only for very short codes [13, Table I]. However, we observe that the complexities of Wolf's algorithm are approximately the same as those of Hwang's for the codes presented in Table I of [13]. Thus, we will compare the performance of our algorithm to that of Wolf.

The simulation results for the $(104,52)$ code for γ_b equal to 5 dB, 6 dB, 7 dB, and 8 dB are given in Table 1. These results were obtained by simulating 35,000 samples for each SNR. Note that the time and space complexities of Wolf's algorithm are proportional to $2^{52} \approx 4.50 \times 10^{15}$.

Table 1: Simulation for the $(104,52)$ code

γ_b	5 dB		6 dB		7 dB		8 dB	
	max	ave	max	ave	max	ave	max	ave
$N(\boldsymbol{r})$	142123	19	2918	1	221	1	0	0
$C(\boldsymbol{r})$	32823	5	519	2	35	2	1	1
$M(\boldsymbol{r})$	13122	4	1912	1	155	1	0	0

where

$N(\boldsymbol{r})$ = the number of nodes visited during the decoding of $\boldsymbol{r}$;

$C(\boldsymbol{r})$ = number of codewords constructed in order to decide on the closest codeword to $\boldsymbol{r}$;

$M(\boldsymbol{r})$ = maximum number of nodes stored on list OPEN during the decoding of $\boldsymbol{r}$;

max = maximum value among 35,000 samples;

ave = average value among 35,000 samples;

$\gamma_b = E_b/N_0$.

Since during simulation no decoding errors occurred for any of the above SNRs, the bit error probability is estimated using the following formula [11]:

$$n_d\sqrt{d_{\min}/(4\pi nk\gamma_b)}\ e^{-(kd_{\min}\gamma_b/n)} \tag{3}$$

where n_d is the number of codewords of Hamming weight $d_{\min}$. The value of n_d was calculated using the results presented in [18]. Table 2 gives an estimate of the bit error probability and coding gain for above SNRs.

Table 2: Bit error probability and coding gain for the (104,52) code

γ_b	5 dB	6 dB	7 dB	8 dB
P_b	$2.028 \times 10^{-10*}$	$5.023 \times 10^{-14*}$	$1.494 \times 10^{-18*}$	$3.079 \times 10^{-24*}$
CG	7.90	8.35	8.80	9.05

P_b = bit error probability;

CG = coding gain (dB);

* Calculate using (3).

We now give the simulation results for the (128,64) code. Since an algebraic decoder which corrects up to 10-bit errors can be constructed for this code, the maximum-likelihood soft-decision decoding algorithm recently proposed in [15] can be implemented. However, in this paper simulation results are given only for very short codes up to length 23. Sub-optimal decoding procedures for this code have been proposed in [10, 3]. Note that the time and space complexities of Wolf's algorithm is proportional to $2^{64} \approx 1.84 \times 10^{19}$.

The simulation results for the (128,64) code for γ_b equal to 5 dB, 6 dB, 7 dB, and 8 dB are given in Table 3. These results were obtained by simulating 35,000 samples for each SNR.

Table 3: Simulation for the $(128, 64)$ code

γ_b	5 dB		6 dB		7 dB		8 dB	
	max	ave	max	ave	max	ave	max	ave
$N(\boldsymbol{r})$	216052	42	13603	2	1143	1	0	0
$C(\boldsymbol{r})$	38219	8	1817	2	91	2	1	1
$M(\boldsymbol{r})$	16626	7	856	1	965	1	0	0

Table 4 gives only an estimate of the bit error probability and coding gain for above SNRs because no decoding error occurred during simulation.

Table 4: Bit error probability and coding gain for the (128,64) code

γ_b	5 dB	6 dB	7 dB	8 dB
P_b	$1.57 \times 10^{-12}*$	$1.71 \times 10^{-16}*$	$1.82 \times 10^{-21}*$	$1.02 \times 10^{-27}*$
CG	8.85	9.22	9.50	9.70

When calculating P_b using (3), the value of $n_d = 243,840$ was taken from [2].

Simulation results for these codes indicate that a drastic reduction on the search space is achieved for the majority of practical communication systems where the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB) [7] even when the algorithm uses a superset of HW.

Simulation results showed that our adaptive decoding algorithm described in this section is at least one order of magnitude more efficient, in time and space, than that proposed in [12], where $S_{C^*} = \{\mathbf{0}\}$ during the entire decoding procedure.

5 Conclusion

In this report we have proposed a novel decoding technique. Simulation results for the above linear block codes attest to the fact that this decoding technique drastically reduced the search space, especially for the majority of practical communication systems where the probability of error is less than 10^{-3} (γ_b greater than 6.8 dB) [7]. For example, the results of Table 3 at 6 dB indicates that for the 35,000 samples tried, this decoding algorithm is approximately 15 orders of magnitude more efficient, in time and space, than Wolf's. Thus, this decoding procedure has not only resulted in an efficient soft-decision decoding algorithm for hitherto intractable linear block codes, but an algorithm which is in fact optimal as well.

We would like to emphasize here the flexibility of this decoding algorithm. For example:

1. It is applicable to any linear block code.

2. It does not require the availability of a hard decision decoder.

3. In order to make it more efficient to decode a particular code, we can design a heuristic function that takes advantage of the specific properties of this code.

4. Any stopping criterion can be easily incorporated into it.

Furthermore, we would like to point out that the algorithm present in this report is suitable for a parallel implementation. One of the reasons is that when calculating $h^{(i)}(m)$ for node m, the algorithm has determined the labels of the path from node m to a node at level $k-2$ that it will follow, so the successors of the nodes in this path can be open simultaneously and processed independently. This will reduce substantially the idle time of processors and the overhead due to processor communication. Thus, we expect a very good speed-up from a parallel version of our algorithm.

This decoding approach will impact both the theoretical and practical branches of coding theory. Theoreticians will be challenged to identify and construct classes of linear codes whose properties maximize the efficiency of this decoding procedure. And practitioners will want to find the most efficient way to implement this algorithm in a fast, single-purpose processor using sequential/parallel structures.

Acknowledgment

The authors would like to thank Elaine Weinman for her invaluable help in the preparation of this manuscript.

Appendix A

In this appendix we will give an example to illustrate our claim that Hwang's algorithm [14] has a fallacy. For the following example his algorithm will fail to start.

Consider the (8,4) extended binary Hamming code generated by

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

Let $\boldsymbol{r}$ be the received vector and $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_7)$ the channel measurement information vector of $\boldsymbol{r}$ [14]. Assume that $\phi_0 < 0, \phi_1 < 0, \phi_2 < 0, \phi_3 > 0, \phi_4 > 0, \phi_5 > 0, \phi_6 > 0, \phi_7 > 0$, and

$$|\phi_0| > |\phi_1| > |\phi_2| > |\phi_4| > |\phi_3| > |\phi_5| > |\phi_6| > |\phi_7|.$$

In order for the first k positions in $\boldsymbol{\phi}$ to have the largest absolute values among all the components of $\boldsymbol{\phi}$, we must swap positions 3 and 4 in $\boldsymbol{\phi}$ and obtain $\boldsymbol{\phi}' = (\phi_0, \phi_1, \phi_2, \phi_4, \phi_3, \phi_5, \phi_6, \phi_7)$. Corresponding to this exchange we have

$$G' = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

which generates code $\boldsymbol{C}'$.

To start the algorithm we must construct a codeword $\boldsymbol{c}'_1 = (c'_{10}, c'_{11}, \ldots, c'_{17})$ of $\boldsymbol{C}'$ such that

$$(-1)^{c'_{10}} \times \phi_0 > 0, (-1)^{c'_{11}} \times \phi_1 > 0, (-1)^{c'_{12}} \times \phi_2 > 0 \text{ and } (-1)^{c'_{13}} \times \phi_4 > 0.$$

Thus, $c'_{10} = 1, c'_{11} = 1, c'_{12} = 1$, and $c'_{13} = 0$. However, $(1,1,1,1,0,0,0,0)$ and $(1,1,1,1,1,1,1,1)$ are the only codewords in $\boldsymbol{C}'$ whose first three bits are ones. Thus, the algorithm will fail in Step 1. The fallacy is in the assumption that any k positions whose components have the largest absolute values among all the components of $\boldsymbol{\phi}$ are linearly independent.

Appendix B

Let $\boldsymbol{\phi} = (\phi_0, \phi_1, \ldots, \phi_{n-1})$ be the received vector. If $|\phi_i| > |\phi_j|$, then we consider that ϕ_i is more "reliable" than ϕ_j, where $|x|$ is the absolute value of x. Let $\boldsymbol{\phi}' = (\phi'_0, \phi'_1, \ldots, \phi'_{n-1})$ be a vector obtained by permuting the positions of $\boldsymbol{\phi}$ such that $|\phi'_i| \geq |\phi'_{i+1}|$ for $0 \leq i < n-1$. The $k \times n$ matrix $\boldsymbol{G}'$ is obtained from $\boldsymbol{G}$ by applying this same permutation to the columns of $\boldsymbol{G}$. In order to give an algorithm to obtain $\boldsymbol{G}^*$, the generator matrix of $\boldsymbol{C}^*$, from $\boldsymbol{G}'$, we first introduce some definitions.

Let $\boldsymbol{A}$ be an $r \times m$ matrix. Given a set $S = \{i_1, i_2, \ldots, i_s\} \subset \{0, 1, 2, \ldots, m-1\}$ we say that S is a sub-information set of $\boldsymbol{A}$ iff the columns of $\boldsymbol{A}$ indexed by $i_1, i_2, \ldots, i_s$ are linearly independent. Furthermore, we define the SW operator. For $0 \leq i, j < m$, $SW(\boldsymbol{A}, i, j)$ is the $r \times m$ matrix obtained from $\boldsymbol{A}$ by swapping columns i and j of $\boldsymbol{A}$.

The following is an algorithm to obtain $\boldsymbol{G}^*$ from $\boldsymbol{G}'$ for $2 \leq k < n$.

```
1. $i \leftarrow 1; j \leftarrow 1; S = \{0\}; \boldsymbol{G}_1^* \leftarrow \boldsymbol{G}'$.
2. If $S \cup \{j\}$ is a sub-information set of $\boldsymbol{G}_1^*$, then $\boldsymbol{G}_1^* \leftarrow SW(\boldsymbol{G}_1^*, i, j)$;
     else
         $j \leftarrow j + 1$;
         go to 2.
3. $S \leftarrow S \cup \{i\}$.
4. If $|S| = k$, then stop;
     else
         $i \leftarrow i + 1$;
         $j \leftarrow j + 1$;
         go to 2.
5. Transform $\boldsymbol{G}_1^*$ into $\boldsymbol{G}^*$ by row operation such that the
       first $k$ columns of $\boldsymbol{G}^*$ form a $k \times k$ identity matrix.
```

The time complexity of the procedure to construct $\boldsymbol{G}^*$ is $O(k^2 \times n)$; however, many of the operations performed during this construction can be done in parallel. In this case, the time complexity becomes $O(k \times n)$.

Appendix C

In this appendix we present an algorithm to calculate $h^{(1)}(m')$ for node m' at level $\ell, -1 \leq \ell < k-1$, whose time complexity is $O(|S_{C^*}| \times n)$.

In the particular case $i = 1$,

$$T(m', \{\boldsymbol{c}^*\}) = \left\{\boldsymbol{v}'_t | d_H(\boldsymbol{v}', \boldsymbol{c}^*) \in HW\right\}, \quad \text{where} \quad \boldsymbol{c}^* \in S_{C^*}.$$

Recall that $\boldsymbol{v}' = (v'_0, v'_1, \dots, v'_\ell, v'_{\ell+1}, \dots, v'_{n-1})$, where $v'_0, v'_1, \dots, v'_\ell$ are the labels of the lowest cost path $\boldsymbol{P}'_{m'}$ from the start node to node m' found so far by the algorithm, and $\boldsymbol{v}'_t = (v'_{\ell+1}, v'_{\ell+2}, \dots, v'_{n-1})$ is a binary $(n-\ell-1)$-tuple.

We now define

$$h(m', \boldsymbol{c}^*) = \min_{\boldsymbol{v}'_t \in T(m', \{\boldsymbol{c}^*\})} \left\{ \sum_{i=\ell+1}^{n-1} \left(\phi^*_i - (-1)^{v'_i}\right)^2 \right\}.$$

Thus, we may write

$$h^{(1)}(m') = \max_{\boldsymbol{c}^* \in S_{C^*}} \left\{ h(m', \boldsymbol{c}^*) \right\}.$$

We now show how we can determine $h(m', \boldsymbol{c}^*)$ from the procedure to find $h(m, \boldsymbol{0})$, where the path P'_m from the starting node to node m can be constructed from $\boldsymbol{c}^* = (c^*_0, c^*_1, \dots, c^*_\ell, c^*_{\ell+1}, \dots, c^*_{n-1})$ and the path $P'_{m'}$.

Let $\boldsymbol{\phi}^*(\boldsymbol{c}^*) = \left((-1)^{c^*_0}\phi^*_0, (-1)^{c^*_1}\phi^*_1, \dots, (-1)^{c^*_\ell}\phi^*_\ell, (-1)^{c^*_{\ell+1}}\phi^*_{\ell+1}, \dots, (-1)^{c^*_{n-1}}\phi^*_{n-1}\right)$ and $v'_0 \oplus c^*_0, v'_1 \oplus c^*_1, \dots, v'_\ell \oplus c^*_\ell$ the labels of P'_m, where $\oplus$ denotes modulo-2 addition. Note that $\boldsymbol{\phi}^*(\boldsymbol{0}) = \boldsymbol{\phi}^*$. We may calculate $h(m', \boldsymbol{c}^*)$ with respect to $\boldsymbol{\phi}^*$ and $h(m, \boldsymbol{0})$ with respect to $\boldsymbol{\phi}^*(\boldsymbol{c}^*)$.

Lemma C1. $h(m', \boldsymbol{c}^*) = h(m, \boldsymbol{0})$.

PROOF. Let $\boldsymbol{v} = \boldsymbol{v}' \oplus \boldsymbol{c}^* = (v_0, v_1, \dots, v_\ell, v_{\ell+1}, \dots, v_{n-1}), \boldsymbol{v}_t = (v_{\ell+1}, v_{\ell+2}, \dots, v_{n-1})$, and $\boldsymbol{c}^*_t = (c^*_{\ell+1}, c^*_{\ell+2}, \dots, c^*_{n-1})$. Recall that for a binary tuple $\boldsymbol{v}'_t = \left(v'_{\ell+1}, v'_{\ell+2}, \dots, v'_{n-1}\right)$ we construct $\boldsymbol{v}' = (v'_0, v'_1, \dots, v'_\ell, v'_{\ell+1}, v'_{\ell+2}, \dots, v'_{n-1})$. First, we note that $\boldsymbol{v}'_t \in T(m', \{\boldsymbol{c}^*\})$

iff $d_H(\boldsymbol{v}', \boldsymbol{c}^*) \in HW$ iff $d_H(\boldsymbol{v}' \oplus \boldsymbol{c}^*, \boldsymbol{0}) \in HW$ iff $d_H(\boldsymbol{v}, \boldsymbol{0}) \in HW$ iff $\boldsymbol{v}_t \in T(m, \{\boldsymbol{0}\})$ iff $\boldsymbol{v}'_t \oplus \boldsymbol{c}^*_t \in T(m, \{\boldsymbol{0}\})$. Thus $\boldsymbol{v}'_t \in T(m', \{\boldsymbol{c}^*\})$ iff $\boldsymbol{v}'_t \oplus \boldsymbol{c}^*_t \in T(m, \{\boldsymbol{0}\})$.

Now we show that $h(m', \boldsymbol{c}^*) = h(m, \boldsymbol{0})$:

$$h(m', \boldsymbol{c}^*) = \min_{\boldsymbol{v}'_t \in T(m', \{\boldsymbol{c}^*\})} \left\{ \sum_{i=\ell+1}^{n-1} \left(\phi^*_i - (-1)^{v'_i}\right)^2 \right\}$$

$$= \min_{\boldsymbol{v}'_t \in T(m', \{\boldsymbol{c}^*\})} \left\{ \sum_{i=\ell+1}^{n-1} \left((-1)^{c^*_i}\phi^*_i - (-1)^{v'_i \oplus c^*_i}\right)^2 \right\}.$$

Since $\boldsymbol{v}'_t \in T(m', \{\boldsymbol{c}^*\}$ iff $\boldsymbol{v}_t = \boldsymbol{v}'_t \oplus \boldsymbol{c}^*_t \in T(m, \{\boldsymbol{0}\})$, we may consider minimization over vectors in $T(m, \{\boldsymbol{0}\})$ instead of in $T(m', \{\boldsymbol{c}^*\})$. Thus

$$h(m', \boldsymbol{c}^*) = \min_{\boldsymbol{v}_t \in T(m, \{\boldsymbol{0}\})} \left\{ \sum_{i=\ell+1}^{n-1} \left((-1)^{c^*_i}\phi^*_i - (-1)^{v_i}\right)^2 \right\} = h(m, \boldsymbol{0}).$$

□

We now present an algorithm to calculate $h(m, \boldsymbol{0})$ whose time complexity is $O(n)$ with respect to any $\boldsymbol{\phi}^*(\boldsymbol{c}^*)$. For easy notation, we denote $h(m, \boldsymbol{0})$ by $h(m)$, $\boldsymbol{\phi}^*(\boldsymbol{c}^*)$ by $\boldsymbol{r}^* = (r^*_0, r^*_1, \ldots, r^*_\ell, r^*_{\ell+1}, \ldots, r^*_{n-1})$, and $T(m, \{\boldsymbol{0}\})$ by $T(m)$.

Let vector $\boldsymbol{u}_\ell = (u_{\ell 0}, u_{\ell 1}, \ldots, u_{\ell(n-\ell-2)})$ be obtained by permuting the positions of $(r^*_{\ell+1}, r^*_{\ell+2}, \ldots, r^*_{n-1})$ in such a manner that $u_{\ell i} \leq u_{\ell(i+1)}$ for $0 \leq i < n - \ell - 2$.

Our algorithm computes $h(m)$ using $\boldsymbol{u}_\ell$ instead of $\boldsymbol{r}^*$. This is possible because of the property

$$h(m) = \min_{\boldsymbol{v}_t \in T(m)} \left\{ \sum_{i=\ell+1}^{n-1} \left(u_{\ell(i-\ell-1)} - (-1)^{v_i}\right)^2 \right\}.$$

This property is easily proved because if $\boldsymbol{v}_t \in T(m)$, then all binary vectors of the same Hamming weight as $\boldsymbol{v}_t$ are contained in $T(m)$ and $\boldsymbol{u}_\ell$ is obtained by applying a permutation π_ℓ to the components of $(r^*_{\ell+1}, r^*_{\ell+2}, \ldots, r^*_{n-1})$.

We now prove some technique lemmas. Consider the set T_w of all binary $(n-\ell-1)$-tuples of Hamming weight w. Furthermore, let $\boldsymbol{v}_p = (v_{p0}, v_{p1}, \ldots, v_{p(w-1)}, v_{pw}, \ldots, v_{p(n-\ell-2)}) \in T_w$, where $v_{pi} = 1, 0 \leq i < w$ and $v_{pi} = 0, w \leq i < n - \ell - 1$.

Lemma C2. *If* $\boldsymbol{v} = (v_0, v_1, \ldots, v_{n-\ell-2}) \in T_w$, *then*

$$\sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_{pi}})^2 \ \leq \ \sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_i})^2 .$$

PROOF.

$$\begin{aligned} D_1 &= \sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_{pi}})^2 - \sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_i})^2 \\ &= 2\sum_{i=0}^{n-\ell-2} \left\{ u_{\ell_i} \left((-1)^{v_i} - (-1)^{v_{pi}}\right) \right\} \end{aligned}$$

Let $S = \{x | v_x = 0 \text{ and } 0 \leq x < w\}$ and $S' = \{x | v_x = 1 \text{ and } w \leq x < n - \ell - 1\}$. Since $\boldsymbol{v}_p = (1, 1, \ldots, 1, 0, 0, \ldots, 0)$ and $W_H(\boldsymbol{v}) = W_H(\boldsymbol{v}_p)$, then $|S| = |S'|$. So $D_1 = 4\left(\sum_{i \in S} u_{\ell i} - \sum_{i \in S'} u_{\ell i}\right) \leq 0$ since $|S| = |S'|$ and $u_{\ell i} \leq u_{\ell j}$, $i \in S$ and $j \in S'$. □

Let $S_\ell = \{x | u_{\ell x} < 0\}$ and $\boldsymbol{v}'_p = (v'_{p0}, v'_{p1}, \ldots, v'_{p(w'-1)}, v'_{pw'}, \ldots, v'_{p(n-\ell-2)}) \in T_{w'}$, where $v'_{pi} = 1,\ 0 \leq i < w'$ and $v'_{pi} = 0,\ w' \leq i < n - \ell - 1$.

Lemma C3. *If $w' < w \leq |S_\ell|$, then*

$$\sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_{pi}})^2 \ < \ \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v'_{pi}}\right)^2 .$$

PROOF.

$$\begin{aligned} D_2 &= \sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_{pi}})^2 \ - \ \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v'_{pi}}\right)^2 \\ &= 4\sum_{i=w'}^{w-1} u_{\ell i} < 0 \text{ since } u_{\ell i} < 0, 0 \leq i < w. \end{aligned}$$

□

Lemma C4. *If $|S_\ell| < w < w'$, then*

$$\sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_{pi}})^2 \leq \sum_{i=0}^{n-\ell-2} \left(u_{\ell i} - (-1)^{v'_{pi}}\right)^2 .$$

The proof of this lemma is similar to that in Lemma C3.

Let $c_0^*, c_1^*, \ldots, c_\ell^*$ be the labels of the path $\boldsymbol{P}'_m$ from the start node to node m at level ℓ found so far by the decoding algorithm. Furthermore, let $c_{\ell+1}^*, c_{\ell+2}^*, \ldots, c_{n-1}^*$ be the labels of a path $\boldsymbol{P}_m$ from node m at level ℓ to the goal node and let $W(\boldsymbol{P}'_m)$ be the number of labels of $\boldsymbol{P}'_m$ whose values are 1. Note that $W(\boldsymbol{P}_m)$ can only have values that belong to the set $Q = \{w_i - W(\boldsymbol{P}'_m) | 0 \leq w_i - W(\boldsymbol{P}'_m) \leq n - \ell - 1 \text{ and } 0 \leq i \leq I\}$.

Let $J \in \{0, 1, \ldots, I\}$ such that $w_J - W(\boldsymbol{P}'_m)$ is the smallest value in Q. Analogously, let $I' \in \{0, 1, \ldots, I\}$ such that $w_{I'} - W(\boldsymbol{P}'_m)$ is the largest value in Q.

By Lemma C2, our algorithm to compute $h(m)$ needs to consider only vectors of the form $\boldsymbol{v}_p = (v_{p0}, v_{p1}, \ldots, v_{p(n-\ell-2)}) = (1, 1, \ldots, 1, 0, 0, \ldots, 0)$ with Hamming weights $w_i - W(\boldsymbol{P}'_m), J \leq i \leq I'$. Furthermore, by Lemmas C3 and C4, we need to consider only the following cases:

Case 1. $|S_\ell| < (w_J - W(\boldsymbol{P}'_m))$. So,

$$h(m) = \sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_{pi}})^2$$

where $W_H(\boldsymbol{v}_p) = w_J - W(\boldsymbol{P}'_m)$.

Case 2. $|S_\ell| \geq (w_{I'} - W(\boldsymbol{P}'_m))$. So,

$$h(m) = \sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_{pi}})^2,$$

where $W_H(\boldsymbol{v}_p) = w_{I'} - W(\boldsymbol{P}'_m)$.

Case 3. $w_{i_1} - W(\boldsymbol{P}'_m) \leq |S_\ell| < w_{i_1+1} - W(\boldsymbol{P}'_m)$. So, $h(m) = \min\{A_1, A_2\}$, where $A_1 = \sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_{pi}})^2$ and $W_H(\boldsymbol{v}_p) = w_{i_1} - W(\boldsymbol{P}'_m)$, and $A_2 = \sum_{i=0}^{n-\ell-2} (u_{\ell i} - (-1)^{v_{pi}})^2$ and $W_H(\boldsymbol{v}_p) = w_{i_1+1} - W(\boldsymbol{P}'_m)$.

Thus, given $\boldsymbol{u}_\ell$ and $|S_\ell|$, the time complexity of computing $h(m)$ is $O(n)$.

Appendix D

D1. Proof of Property 1

Let node m_2 at level ℓ be an immediate successor of node m_1. Furthermore, let c_ℓ^* be the label of the arc from node m_1 to node m_2 and $c(m_1, m_2) = \left(\phi_\ell^* - (-1)^{c_\ell^*}\right)^2$. We now prove that $h^{(i)}(m_1) \leq h^{(i)}(m_2) + c(m_1, m_2)$.

(a) $\ell < k-1$. Let $Y_i \in P_i(S_{C^*})$. Furthermore, let $\boldsymbol{v}_t' = (v_{\ell+1}', v_{\ell+2}', \ldots, v_{n-1}') \in T(m_2, Y_i)$ such that

$$\min_{\boldsymbol{v}_t \in T(m_2,Y_i)} \left\{ \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \right\} = \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v_i'}\right)^2 .$$

Since $\boldsymbol{v}_t' \in T(m_2, Y_i)$, then $\left(c_\ell^*, v_{\ell+1}', v_{\ell+2}', \ldots, v_{n-1}'\right) \in T(m_1, Y_i)$. Thus

$$\min_{\boldsymbol{v}_t \in T(m_2,Y_i)} \left\{ \sum_{i=\ell+1}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \right\} + c(m_1, m_2) \geq \min_{\boldsymbol{v}_t \in T(m_1,Y_i)} \left\{ \sum_{i=\ell}^{n-1} (\phi_i^* - (-1)^{v_i})^2 \right\} .$$

Thus, $h^{(i)}(m_2) + c(m_1, m_2) \geq h^{(i)}(m_1)$.

(b) $\ell = k-1$. $h^{(i)}(m_1) \leq h^*(m_1)$ and $h^{(i)}(m_2) = h^*(m_2)$. Since $h^*(m_1) - c(m_1, m_2) \leq h^*(m_2)$, then $h^{(i)}(m_1) \leq h^*(m_2) + c(m_1, m_2) = h^{(i)}(m_2) + c(m_1, m_2)$.

(c) $\ell > k-1$. $h^{(i)}(m_1) = h^*(m_1)$ and $h^{(i)}(m_2) = h^*(m_2)$. Since $h^*(m_1) - c(m_1, m_2) = h^*(m_2)$, then $h^{(i)}(m_1) = h^{(i)}(m_2) + c(m_1, m_2)$.

D2. Proof of Property 2

Consider node m_ℓ at level ℓ, $-1 \leq \ell < k-2$. Furthermore, let $h^{(i)}(m_\ell) = \sum_{i=\ell+1}^{n-1} \left(\phi_i^* - (-1)^{v_i'}\right)^2$ where $(v_{\ell+1}', v_{\ell+2}', \ldots, v_{n-1}') \in T(m_\ell, Z)$ for some $Z \in P_i(S_{C^*})$. Now consider the path $\overline{\boldsymbol{P}}_{m_\ell} = (m_\ell, m_{\ell+1}, \ldots, m_{k-2})$ from node m_ℓ to node m_{k-2} at level $k-2$ whose labels are $v_{\ell+1}', v_{\ell+2}', \ldots, v_{k-2}'$. We now show that if $m_{\ell+1}$ is a node in this path at level $\ell+1$, then $f(m_\ell) = f(m_{\ell+1})$.

By definition $f(m_\ell) = g(m_\ell) + h^{(i)}(m_\ell) = g(m_\ell) + \left(\phi^*_{\ell+1} - (-1)^{v'_{\ell+1}}\right)^2 + \sum_{i=\ell+2}^{n-1} \left(\phi^*_i - (-1)^{v'_i}\right)^2 = g(m_{\ell+1}) + \sum_{i=\ell+2}^{n-1} \left(\phi^*_i - (-1)^{v'_i}\right)$. Since $(v'_{\ell+2}, v'_{\ell+3}, \ldots, v'_{n-1}) \in T(m_{\ell+1}, Z)$, then

$$\sum_{i=\ell+2}^{n-1} \left(\phi^*_i - (-1)^{v'_i}\right)^2$$

$$= \min_{\boldsymbol{v}_t \in T(m_{\ell+1}, z)} \left\{ \sum_{i=\ell+2}^{n-1} (\phi^*_i - (-1)^{v_i})^2 \right\},$$

otherwise

$$h^{(i)}(m_\ell) > \min_{\boldsymbol{v}_t \in T(m_\ell, Z)} \left\{ \sum_{i=\ell+1}^{n-1} (\phi^*_i - (-1)^{v_i})^2 \right\}.$$

Analogously, we can conclude that

$$h^{(i)}(m_{\ell+1}) = \sum_{i=\ell+2}^{n-1} \left(\phi^*_i - (-1)^{v'_i}\right)^2.$$

References

[1] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. on Information Theory*, pp. 284–287, March 1974.

[2] L. D. Baumert and L. R. Welch, "Minimum-Weight Codewords in the (128,64) BCH Code," DSN Progress Report 42–42, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, September and October 1977.

[3] L. D. Baumert and R. J. McEliece, "Soft Decision Decoding of Block Codes," DSN Progress Report 42–47, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, July and August 1978.

[4] E. R. Berlekamp, *Algebraic Coding Theory.* New York, NY: McGraw-Hill Book Co., 1968.

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

[6] R.W. D. Booth, M. A. Herro, and G. Solomon, "Convolutional Coding Techniques for Certain Quadratic Residue Codes," in *Proc. 1975 Int. Telemetering Conf.*, pp. 168–177, 1975.

[7] G. C. Clark, Jr., and J. B. Cain, *Error-Correction Coding for Digital Communications.* New York, NY: Plenum Press, 1981.

[8] J. H. Conway and N. J. A. Sloane, "Soft Decoding Techniques for Codes and Lattices, Including the Golay Code and the Leech Lattice," *IEEE Tran. on Information Theory*, pp. 41–50, January 1986.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms.* Cambridge, MA: The MIT Press, 1991.

[10] B. G. Dorsch, "A Decoding Algorithm for Binary Block Codes and J-ary Output Channels," *IEEE Trans. Information Theory*, pp. 391–394, May 1974.

[11] G. D. Forney, Jr., *Concatenated Codes.* Cambridge, MA: The M.I.T. Press, 1966.

[12] Y. S. Han, C. R. P. Hartmann, and C-C. Chen, "Efficient Maximum-Likelihood Soft-Decision Decoding of Linear Block Codes Using Algorithm A^*," Technical Report SU-CIS-91-42, School of Computer and Information Science, Syracuse University, Syracuse, NY 13244, December 1991.

[13] T.-Y. Hwang, "Decoding Linear Block Codes for Minimizing Word Error Rate," *IEEE Trans. on Information Theory*, pp. 733–737, November 1979.

[14] T.-Y. Hwang, "Efficient Optimal Decoding of Linear Block Codes," *IEEE Trans. on Information Theory*, pp. 603–606, September 1980.

[15] T. Kancko, T. Nishijima, H. Inazumi, and S. Hirasawa, "An Efficient Maximum-Likelihood-Decoding Algorithm for Linear Block Codes with Algebraic Decoder," submitted for publication to *IEEE Trans. Information Theory.*

[16] L. B. Levitin, M. Naidjate, and C. R. P. Hartmann, "Generalized Identity-Guards Algorithm for Minimum Distance Decoding of Group Codes in Metric Space," presented at the 1990 IEEE International Symposium on Information Theory, San Diego, CA, January 1990.

[17] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes.* New York, NY: Elsevier Science Publishing Company, Inc., 1977.

[18] C. L. Mallows and N. J. A. Sloane, "An Upper Bound for Self-Dual Codes," *Information and Control*, 22, pp. 188–200, 1973.

[19] M. Naidjate, "Generalized Minimum Distance Decoding Algorithms for Group Codes in Metric Spaces," Ph.D. dissertation, Boston University, Boston, MA, 1991.

[20] N. J. Nilsson, *Principle of Artificial Intelligence.* Palo Alto, CA: Tioga Publishing Co., 1980.

[21] I. S. Reed, T. K. Truong, X. Chen, and X. Yin, "The Algebraic Decoding of the (41,21,9) Quadratic Residue Code," *IEEE Trans. on Information Theory*, pp. 974–986, May 1992.

[22] G. Solomon and H. C. A. van Tilborg, "A Connection Between Block and Convolutional Codes," *SIAM J. Appl. Math.*, pp. 358–369, 1979.

[23] D. J. Taipale and M. B. Pursley, "An Improvement to Generalized-Minimum-Distance Decoding," *IEEE Trans. on Information Theory*, pp. 167–172, January 1991.

[24] A. J. Viterbi, "Error Bound for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. on Information Theory*, pp. 260–269, April 1967.

[25] J. K. Wolf, "Efficient Maximum Likelihood Decoding of Linear Block Codes Using a Trellis," *IEEE Trans. on Information Theory*, pp. 76–80, January 1978.